LOWER BOUND RESOURCE REQUIREMENTS FOR MACHINE INTELLIGENCE

Timur Gilmanov

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the School of Informatics, Computing, and Engineering
Indiana University
November 2018

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Thomas Sterling, Ph.D.

David Leake, Ph.D.

Sandra Kübler, Ph.D.

Martin Swany, Ph.D.

October 16, 2018

To my parents, Anvar and Liubov Gilmanov, who were
the first to introduce me to the process of learning.

Acknowledgements

This work would have not materialized without the help of my colleagues, family, and friends all of whom I am eternally grateful to.

First, I would like to thank my academic advisor, Thomas Sterling for setting an example of excellence. It was his accurate guidance, stimulating discussions, encouragement, and support over the course of this program that helped me to get to the finish line. I would like to thank my research committee members: Sandra Kübler, David Leake, and Martin Swany for their guidance, suggestions, and feedback on my thesis.

I am very grateful to Matthew Anderson and Maciej Brodowicz for their mentorship, guidance, advice, and friendship over the years that we have been working together. In addition to collaborations on a number of projects and publications outside of the scope of this thesis, their work and advice have significantly helped to shape the outcome of this research effort.

I would very much like to thank all of my collaborators and friends at the School of Informatics, Computing, and Engineering (SICE), and the former Center for Research in Extreme Scale Technologies (CREST), including Sarah Carroll, Kim Rodman, Laura Pettit, Amanda Upshaw, Rebecca Schmitt, Patricia Reyes-Cooksey, Cathy McGregor Foster, Regina Helton, and Julie Overfield. I am particularly grateful to Kim and Sarah for their support, friendship, and for always being there for me since the first day that we have met.

I would like to thank my fellow graduate students, collaborators, and friends at the SMART laboratory and formerly CREST, including Daniel Kogler, Larisse Voufo, Bibrak Chandio, Prateek Srivastava, Lucas Brasilino, Jesun Firoz, Abhishek Kulkarni, Marcin Zalewski, and the rest of the crew for the great collaborative environment, engaging research activities, and fun times during our years at the graduate school.

I would also like to thank all of the Computational Linguistics students, faculty, and collaborators from the Department of Linguistics at Indiana University (IU), including Markus Dickinson, Alex Rudnick, Can Liu, Eric Baucom, Levi King, Olga Scrivner, and Evgeny Kim for numerous

collaborations, informative CLindDing seminars, and extracurricular activities. I consider myself lucky to have discovered this program early on in my days at IU.

I would also like to thank my former colleagues and external collaborators: Hartmut Kaiser, Chris Michael, Dylan Stark, and Chirag Dekate. Hartmut was the first to introduce me to the world of C++.

I would like to thank all the IU professors that taught various courses during my years in the program. Most importantly, I would like to mention Kris Hauser, Predrag Radivojac, Michael Trosset, and Gregory Rawlins.

Another big thank you is due to the wonderful city of Bloomington, IN, its community, and all of my friends there. Most notably, I am thankful to Pavel Anaschenko, Aleksey Kolpakov, Vlad Bereznyuk, Ala Simonchyk, Elena Doludenko, Veronika Trotter, and Eric Morales.

I am very thankful to IU CREST and the Departments of Computer Science and Intelligent Systems Engineering for the financial support during my time in graduate school.

I owe a great thank you to my Alma mater, the former Kazan State University (KSU)—presently named Kazan (Volga region) Federal University (KPFU)—for providing the necessary foundation and knowledge that allowed me to undertake this work. In particular, I am very grateful to Professor Nail Bukharaev from the Department of Programming Technologies, who had done an incredible job of introducing programming languages and computer science to me during the freshman year at KSU. I am also very thankful to my undergraduate academic advisor at KSU, Rafail Dautov.

I am very grateful to my friends and fellow undergraduate students from KSU, Ilya Dolgov and Alexander Shlyannikov, for everything we have been working on together over the years, for our friendship and wonderful adventures. I am very thankful to my friend from Kazan, Rushan Sibgatullin, who always believed in my work and inspired me since my first days at graduate school, and who will be missed dearly.

Finally, I give my sincere thanks to my family: Anvar, Liubov, and Ildar Gilmanov. Your encouragement, faith in me, and help throughout these years have been invaluable.

Most importantly, I would like to thank my wife, Anna Lysiuk, whose support has become a crucial part of my success in this endeavor. Thank you for always believing in me.

This work has been partially supported by NSF awards 1538642 and 1440396.

Timur Gilmanov

LOWER BOUND RESOURCE REQUIREMENTS FOR MACHINE INTELLIGENCE

Recent advancements in technology and the field of artificial intelligence provide a platform for new applications in a wide range of areas, including healthcare, engineering, vision, and natural language processing, that would be considered unattainable one or two decades ago. With the expected compound annual growth rate of 50% during the years of 2017–2021, the field of global artificial intelligence is set to observe increases in computational complexities and amounts of sensor data processed.

In spite of the advancements in the field, truly intelligent machine behavior operating in real time is yet an unachieved milestone. First, in order to quantify such behavior, a definition of machine intelligence would be required, which has not been agreed upon by the community at large. Second, delivering full machine intelligence, as defined in this work, is beyond the scope of today's cutting-edge high-performance computing machines.

One important aspect of machine intelligent systems is resource requirements and the limitations that today's and future machines could impose on such systems. The goal of this research effort is to provide an estimate on the lower bound resource requirements for machine intelligence. A working definition of machine intelligence for purposes of this research is provided, along with definitions of an abstract architecture, workflow, and performance model. Combined together, these tools allow an estimate on resource requirements for problems of machine intelligence, and provide an estimate of such requirements in the future.

Contents

ACS	Active Context Stack
AI	Artificial Intelligence
AMA	Autonomous Moving Agent
AOS	Active Objective Function Stack
ASIC	Application-Specific Integrated Circuit
ASR	Automatic Speech Recognition
CRIS	Cognitive Real-time Interactive System
FLOP	Floating Point Operation
FLOPS	Floating Point Operations per Second
GPU	Graphics Processing Unit
HPC	High Performance Computing
MCP	Master Control Program
MI	Machine Intelligence
ML	Machine Learning
NLP	Natural Language Processing
OF	Objective Function
Ops	Operations
PDDL	Planning Domain Definition Language
RBF	Radial Basis Functions
TEPS	Traversed Edges per Second

1 Introduction

The field of Artificial Intelligence (AI) has been on the rise in the past two decades, with more than nine times the increase of the number of academic papers published since 1996, 14 times the increase of startups that work on some sorts of AI systems, and a little under five times the number of jobs that require AI skills since 2013, among many other examples of its considerable growth [**1**]. Recent progress in technology allows for an extensive range of applications in the field, varying from facial and speech recognition to self-driving vehicles and personal assistants.

With latest advancements in the field of Artificial general intelligence, including software and hardware, little attention has been paid to questions of estimating how much computational resources, such as computer performance, power, memory, and network considerations would be required to support these applications. Software companies that deliver systems of facial recognition, automatic speech recognition, and autonomous moving vehicles, to name a few, do not normally reveal the number of hours, or computational complexities, that are associated with training required for such systems. Although research questions concerned with estimating requirements of a few sample applications have been considered in the past [**32**, **37**, **56**], it is not a usual practice to include time to solution or resource requirements considerations (e.g. complexity analysis for Operations (Ops), Floating Point Operations per Second (FLOPS), or main memory requirements) along with newly introduced approaches in the field. As a result, an attempt to perform a detailed investigation of resource requirements for the problems in the area of AI has not been delivered to date. An approach that would fill this gap and present insights on the question of AI system resource needs could provide an understanding about the types and scale of the hardware systems positioned to support systems with intelligent behavior. Such understanding will allow to make

better decisions about the types and amounts of hardware that would be truly needed for future systems, applications and algorithms in the field.

In contrast to Artificial Intelligence, a term Machine Intelligence (MI) is introduced as a notion of a class of systems with intelligent behavior that involve an entire workflow when processing the tasks or external stimuli directed at them. Machine Intelligence is defined as a finite algorithm that i) dynamically models an external environment of an agent that is subjected to such environment, ii) performs processing of tasks by satisfying a set of objectives and axioms, iii) is self-aware by including a model of itself in the representation of external environment, iv) processes the tasks in real time. A detailed definition of MI along with its description is provided later in Chapter 3.

The present work is an attempt to explore in a practical way the question of resource estimation for MI.

1.1. Challenges

The field of Artificial Intelligence today is comprised of a wide variety of computational approaches to solving problems ranging from mobile robot navigation and autonomous self-driving cars to personal assistants, logic game players and dialogue systems. Some of the past advancements were believed at various points in time to have succeeded to pass versions of the Turing test or even to exceed the human intelligence capabilities in particular domains. Whenever such claims were made, the systems possessing the intelligent behavior were special-case systems that were tailored for executing a specific task. Some examples of such systems are Joseph Weizenbaum's ELIZA [**192**], Deep Blue chess computer [**29**], the IBM Watson Jeopardy competition [**58**], object detection of ImageNet image collection [**85**], and Google's AlphaGo [**170**].

Even though the approaches mentioned were special-case systems, in contrast with the general AI type of systems, the progress made in these various fields is of great importance, as advancements in these areas will be incorporated in general AI systems of the future. These approaches using Machine Learning (ML) algorithms showed some significant accuracy prediction improvements in speech and image recognition, however they still often lack the level of perception inherent to human intelligence. A good example of this is a study named "One pixel attack for fooling deep neural networks" [**180**], where test data had one insignificant alternation—a change of color in only one pixel. The results of that alternation were astonishing: some of the test data images

were completely misidentified by the system with levels of confidence exceeding 99 percent for some cases. Another example of a shortcoming of an approach in ML, which in this case is related to neural networks, is described in [**45**]. The authors show how a bias is formed towards one meaning or another depending on the order when a neural network is learning a semantic representations of homonyms (two or more words that are the same in spelling but have different meanings).

Some of the biggest challenges for this research are the same challenges that are true for the broader field of AI: 1) it is not yet clear what generic MI/AI truly is; 2) for some cases, the AI problems are unsolvable, or undecidable, posessing exponential algorithmic complexities; 3) finally, defining relevant characteristics of the experiment is a challenge. Special consideration is provided in later Chapters in regards with describing the metrics and requirements parameters used in order to conduct this study.

These challenges are faced by carefully setting up the experiments and providing a working definition of what the term "Machine Intelligence" means.

1.2. Goals and Objectives

The goal of this research is to provide lower bound resource requirements estimates for MI. In order to satisfy this goal, a few objectives need to be completed:

1) Provide a working definition for Machine Intelligence.
2) Introduce a model that describes applications of MI. The model comprises three independent components: the abstract architecture, the workflow, and the performance model. All three are used in order to define the operational behavior and to measure the performance of Cognitive Real-time Interactive System (CRIS)—a system introduced in this work that is used as a foundation to verify the concepts described in this research.
3) Define a set of experiments that will be driving the abstract architecture and its implementation, CRIS.
4) Collect the results of the experiments conducted and perform analysis of these results.

1.3. Strategy

The strategy for achieving the goals and objective of this research study are outlined below.

1.3.1. Resource requirements. The resource requirements will be estimated using one metric, the FLOPS, and one requirement parameter—the amount of main memory of a system that the computations are performed on.

1.3.2. Architecture of Machine Intelligence. The architecture of MI is defined in hierarchical modular fashion and can be used to provide a direct implementation of a system. That approach was chosen in order to provide an implementation CRIS.

1.3.3. Workflow. The workflow defines how the elements of abstract architecture interact with each other every iteration of CRIS's execution.

1.3.4. Performance Model. The performance model will be used in order to provide estimates of resource requirements characteristics for both, today's problems of MI, as well as problems of the near future, given that the information about the corresponding algorithmic complexities as well as amounts of data processed are known or can be estimated.

1.3.5. Machine Intelligence External Drivers. The external drivers for MI will be used in order to provide specific examples of the Workflow for MI in action. The drivers will vary in the types of activities they represent, including the Winograd's blocks world as a simple analogy of the more complicated dialogue system, the Autonomous Moving Agent (AMA), and the 3-D facial recognition system.

1.3.6. Experiments. The experiments will include execution of the external drivers subjected to the relevant parts of the workflow, as well as collecting and analyzing data associated with performance and resource requirements estimation.

1.4. Thesis Statement

Resource requirements for the problems of Machine Intelligence can be monitored and predicted for a large number of workflows.

1.5. Thesis Organization

This thesis is organized as follows. This Chapter provides an overview of current problems in the fields of Artificial and Machine Intelligence, as well as recognizes the importance of resource requirements estimates for the field, and outlines an approach of how this research is conducted.

Chapter 2 presents a survey of prior work in the fields of AI, and High Performance Computing (HPC).

Chapter 3 examines views of various schools of thought on a question of what Intelligence, and Machine Intelligence are. A summary of a broad range of definitions is provided. The rest of the Chapter is dedicated to providing a working definition of Machine Intelligence that is used throughout this research.

Chapter 4 introduces the first piece of a model of MI, named Abstract Architecture. It describes a hierarchy of building blocks of an example of a system for MI, CRIS.

Next, Chapter 5 introduces a generic Workflow that shows how various elements of the Abstract Architecture interact with each other in CRIS.

Chapter 6 introduces the final constituent of the model for MI—the Performance Model. The performance model provides an approach to estimation of MI algorithms of MI. The model incorporates existing complexity analysis for applications and algorithms in MI. Two metrics, used to characterize resource requirements for MI are introduced and discussed in this Chapter. Finally, a few additional metrics that may be useful for future estimations are introduced.

Chapter 7 describes the experimental setup for three independent external drivers that are used as example applications of MI to be executed on top of the Workflow. The considered drivers are the Winograd's Blocks World, AMA, and 3-D Facial Recognition.

Chapter 8 provides an overview of results obtained for the three driver applications mentioned above.

Next, Chapter 9 includes an analysis based on the obtained results. It provides an estimate on the lower bound resource requirements of today's applications of MI. The Chapter is concluded by an analysis of future trends for the field of MI, as well as the future trends of the performance growth for HPC systems that may be required to support the execution of such applications in real time.

Lastly, Chapter 10 summarizes the results of present work and provides conclusions.

2

Survey of prior research

This chapter provides an overview of work done to date in areas of Machine Intelligence (MI) and Artificial Intelligence (AI) and any applicable attempts of measuring intelligent systems. Prior research overview presented in this chapter benefits and expands the horizons of two immensely important areas of Computer Science—AI and High Performance Computing (HPC). Both of these major fields include a myriad of subfields with a vast number and variety of applications. Some examples of such applications, relevant to this work, include Natural Language Processing (NLP), vision and image processing, Automatic Speech Recognition (ASR), intelligent agents and robotics, among many other applications in Machine Learning (ML) and AI, and virtually all aspects of engineering and numerical simulations, including applications in physics, geology, meteorology, and biology, as well as any other computationally demanding fields of applications in the HPC community. It is almost impossible for sophisticated modern machine intelligent agents to not utilize the HPC systems available today for improved performance, with an exception of a few types of computation, where the individual aspects, such as planning, ASR, or factual inferences are involved. These particular examples' computational resource requirements can be satisfied by modern mobile device architectures due to the fact that the maximal performance (R_{max}) of today's mobile device processors can reach up to ≈ 5 GFLOPS [**135**], which is comparable to performance of supercomputers in early to mid 1990s. Supporting real-time computation and quick processing times, however, requires more capable hardware systems. Algorithms in decision making, deep learning, speech, text, and vision processing have been benefitting from HPC approaches in the recent past.

This research effort benefits from some ideas of the prior state of the art described below, including approaches to architectures of AI, as well as specific applications of AI, including T.

Winograd's BLOCKS world, Autonomous Moving Agent (AMA), and facial recognition, as well as general-purpose HPC systems that were used in order to conduct the required experiments. An overview of the main challenges and advancements in both fields is provided below.

2.1. Artificial Intelligence

Dialogue systems or conversational agents have been a focus of research for the last few decades. The gap between humans and personal assistants with AI capabilities had been substantially narrowing throughout recent years, especially with research and engineering advancements in the last decade, when technologies like Apple's Siri [**172**] and Google's ASR got introduced [**10**, **71**]. In spite of substantial achievements in this field, performance of modern ASR systems is still far from desirable [**6**]. Such systems are doing well when subjected to ideal conditions, including noise-free environments, or in scenarios substantial training data is available for the language in question (i.e. more popular languages like English vs less popular "low-resource" languages). If, however, these systems are subjected to real-world conditions, the performance drops significantly, oftentimes with the outcome of systems failing completely. Dialogue systems are directly related to the work described in this thesis. They are predicted to constitute a substantial fraction of machine intelligent agents and will play a crucial role in how such agents interact with external entities.

2.1.1. Dialogue systems. One of the earliest examples of dialogue systems, presented by Joseph Wizenbaum in 1966, named ELIZA, is an example of a primitive (by modern standards) NLU application [**192**]. Although based on simple principles, ELIZA was taken seriously by some users, who thought that the system was capable of intelligent cognitive process during a dialogue with a human. ELIZA was implemented using straightforward pattern matching and parsing and if certain users requests exceeded its knowledge base, it would return a generic response by restructuring a statement into a question. Directions to the system on how to interact with users were provided by scripts. Such scripts contained collections of keywords and transformation rules. The most famous script, DOCTOR, imitated behavior of a psychotherapist. Although more complex systems were later introduced and were considered successful, ELIZA remains a milestone as the first attempt to introduce such way of human-machine interaction.

Another important milestone that brought the dialogue systems to a whole new level was T. Winograd's SHRDLU introduced in 1968–1970 at MIT [**196–198**]. SHRLDU was an early Natural Language Understanding (NLU) program that was involved into a conversation with a user. The world of this system consisted of blocks that vary in shape and color. A user could request for certain arrangements of the blocks in the world by providing natural language commands or questions. The program would analyze the obtained commands via a terminal in form of typed text, interpret them, attempt to come up with a plan of actions and execute any acceptable plan. What made SHRLDU successful was a combination of 3 ideas that combined together were considered a great breakthrough. One was the small "size" of the world, which could be described by a really limited vocabulary, including words (and their closest synonyms) like "block", "box", "pyramid", "object", various colors, including "red", "green", "blue", as well as sizes, such as "large" and "small". The second factor that contributed to the system's success was the memory functionality. A user could ask additional questions about the actions performed in the past, like "Have you picked a red pyramid up since we began?". The system kept track of all its prior actions and was capable of providing an answer. The memory feature also allowed the system to provide information about the semantics of world's objects, such as "Can I put a block on top of another block?". The system would consult its previous history to make a decision of whether such action was possible. Finally, the system was capable of identifying references to earlier part of discourse. A question "Why have you picked it up?" would initiate analysis that would determine that a user is probably referring to an object which was discussed most recently. Unfortunately, the initial excitement about Winograd's system did not materialize into a true AI system capable of reasoning and maintaining a dialogue without being limited to the blocks world.

Later, during the 1970s a lot of efforts were concentrated around conceptual ontologies, which structured information about real-world into data understandable to machines. Examples of such work include MARGIE by Schank and Abelson [**166**], SAM [**43**] by Cullingford, and Politics [**31**] by Carbonell.

It was predicted by the AI community throughout the period of 1950s to 1970s that the problem of creating systems of machine intelligence would be solved in the near future (within a decade or a generation) [**134**, **171**]. These results, however, were not achieved largely due to the misbelief of the community that microworld systems would be easy to extend onto the real-world arbitrary

scenarios by applying the same underlying principles. A problem that such approaches were facing is known as "combinatorial explosion" [**120**], where the algorithmic complexities were impossible for the machines to deal with in order to provide adequate solutions within given time constraints. Modern dialogue systems are still facing many issues that the designers of early systems were challenged by. Among others, these include: 1) lack of exploration, when a system has to be tightly controlled by a user; 2) integrating learned and designed behavior, when even most interactive systems today have to stay relatively close to a skeleton scenario in dialogue; 3) multifunctional behavior, when systems are required to perform multiple actions in parallel, as Ward et al. are suggesting [**191**].

2.1.2. Machine Learning. Another large class of AI applications is associated with Machine Learning approaches that are applied to a variety of domains, including audio, video and text processing, classification, clustering and categorization among many others [**17**, **70**, **168**, **184**].

During the 1980s an idea of back-propagation learning [**54**,**164**] initially introduced in 1969 [**26**] had found its application to fields of computer science and psychology. There had been a debate in the AI community that this new so-called connectionist models of intelligent systems would replace symbolic computing approaches. Although this question is still not answered to date, many believe that the two approaches should be coupled for machine intelligence to be achieved.

Throughout the 1970s a lot of efforts in concentrated around the automatic speech recognition [**41**, **113**, **136**]. A lot of various approaches were tried, but best results throughout the 1980s and 1990s were achieved by Hidden Markov Models (HMM) [**159**, **160**]. There were two reasons why HMMs were performing so well, compared to other approaches: 1) this model was built on rigorous mathematical theory, allowing to utilize the experience gained from many other fields 2) this approach used large amounts of real recorded data, which allowed the system to "learn" based on examples, thus making a system robust and flexible to specific data it is working on.

A special class of Machine Learning, Neural Networks [**82**, **83**, **92**, **107**], and especially their recent successful results in Deep Learning [**7**, **146**, **167**] take a special place in today's research on tasks associated with both supervised and unsupervised approaches to data classification, clustering and pattern recognition.

By the mid-1990s a group of researchers inspired by advancements in various domains of AI made attempts to develop the "whole" machine-intelligent architectures that were aimed to

deliver architectures of general AI systems. One most notable example of a complete agent architecture, named SOAR, was proposed by Newell, Laird, and Rosenbloom [**110**, **111**]. These efforts were continued later with suggestions for general AI architectures by Looks et al [**122**] and Bonasso et al [**154**]. The proposed ideas on whole-agent architectures are directly related to research discussed in this thesis, as a model for machine intelligence introduced later in Chapter 4.

Most recent advancements in the area of machine learning are based on using massive amounts of training data available on the Web today. Domains that benefit from this approach include text [**99**] and image [**84**] processing. This approach suggests that the performance of systems utilizing machine learning depends largely on the amount of training data available, and less so on specific algorithm.

2.1.3. Planning. Planning, as an important attribute of MI systems, finds its place in recent research advancements with a number of problems considered in [**20**, **44**, **86**, **161**]. The original STRIPS planning language [**60**], introduced in 1971, has been the standard approach for solving classical planning problems—the ones that operate in fully observable, deterministic, static environments with one agent. However, more sophisticated planning approaches were introduced later and have been applied to a range of applications, including the real-world planners utilizing domain knowledge [**88**, **195**].

2.2. Programming models

LISP is a family of computer programming languages. Initially introduced in 1958, LISP is one of the oldest high-level programming, dialects of which, including Scheme and Common LISP, are still used today. Although initially created for easy expression of mathematical notation, this language quickly became the language of choice in the AI community. During its early years, a number of systems were expressing their AI capability by means of LISP, including T. Winograd's SHRDLU Micro Planner [**196**–**198**], early work by McCarthy named Advice Taker [**127**], a general purpose algebra system based on symbolic computing Macsyma [**125**], a comprehensive ontology assembly project CYC [**118**] and many others. The reason LISP found such a substantial support in AI community is that it naturally and straightforwardly supports ways for symbolic computing.

As mentioned earlier in this Chapter, symbolic computing was considered the main approach to achieve machine intelligence in early days of AI.

PROLOG is another general-purpose programming language that is associated with AI [**39**]. The implementation of this language is supported by mechanisms, including pattern-matching, tree-based data structuring and automatic backtracking. Writing Prolog programs involves defining rules and facts. For example, a system of objects and their spatial relationships could be easily defined. A user can then inquire such program about certain relationships between objects in question. Prolog was also favored by the AI community due to its expressiveness and support for symbolic reasoning, database operations and language parsing capabilities. Among various projects utilizing Prolog is the Fifth Generation project [**57**, **142**].

Maple is another example of symbolic and numeric computing environment [**34**, **35**]. Unlike previous programming languages, the MAPLE's main kernel is implemented in C. Users can enter requests in traditional mathematical notation.

Macsyma (Project MAC's SYmbolic MAnipulator) is a general purpose computer algebra system for manipulation of symbolic and numerical expressions, including integration, differentiation, Taylor series, etc. [**141**]. The development on this system was ongoing from 1968 to 1982 at MIT's Project MAC. In 1982 Macsyma was licensed to company Symbolics and became commercial software. A modified version of original software named Maxima is still maintained today.

2.3. High Performance Computing

The early days of supercomputers go back to 1965, when first Control Data Corporation's CDC 6600 model was delivered by Seymour Cray and his colleagues [**62**]. At the time of its delivery, the CDC 6600 was outperforming every other solution on the market by roughly ten times with the attained performance of 500 KFLOPS utilizing late-model FORTRAN compilers with 1 MFLOPS peak performance [**176**].

Later, in 1980s, the Connection Machines, a series of supercomputers designed by D. Hillis, intended for applications in AI were introduced [**89**]. Some examples of applications that were designed with ideas of parallelism due to introduction of Connection Machines included natural language interpretation [**189**] along with computer vision, object recognition, machine learning and information retrieval [**186**]. The theoretical performance of a CM-2 machine was estimated to

be 2.5 GFLOPS [**150**]. The later version of Connection Machines, CM-5, although not a dedicated AI machine, was the second fastest system in the November 1993 TOP500 [**179**] list, running a benchmark on 1024 cores with $r_{max} = 59.7$ GFLOPS.

Another important area of research were Lisp Machines [**53**, **137**], built and sold by several companies in the 1980s, including Symbolics, Lisp Machines Incorporated, Texas Instruments and Xerox. These machines were designed with a purpose of having a special hardware, optimized to work with the programming language Lisp, and the machines were supposed to provide supports computation in the area of AI. With the onset of "AI winter" and the technological advancements the Lisp Machines were soon replaced by Desktop PCs that could run the Lisp programs faster and not requiring any specialized hardware.

At approximately the same time, an attempt of Japan's government was made to create a computer using massively parallel computing [**57**, **142**]. This initiative begun in 1982. The idea of introducing the 5th generation was in turning away from microprocessors (the 4th generation) towards massive numbers of CPUs combined together for added performance. The project was not successful due to the same reasons as with Lisp Machines.

Examples of research activities in Massively Parallel AI and Parallel Computing AI applications are presented in [**22**, **47**, **101**, **108**].

Finally IBM's recent advancements to develop electronic neuromorphic machine technology that scales to biological levels, SYNAPSE, are presented in the following publications [**8**, **130**, **131**].

2.4. Systems for Artificial Intelligence

There are a few examples of hardware systems that were developed to support symbolic computing in the past. A few of such systems, namely LISP machines and CM-2, are discussed in Section 2.3 of this Chapter. Other important systems that enabled execution of problems in AI are provided below.

In 1950 Marvin Minsky and Dean Edmonds built the first artificial neural network computer that modeled the behavior of a rat in the search for food. This system was called Stochastic Neural Analog Reinforcement Computer (SNARC). It was designed in a way where the synapses would adjust their weight based on successful fulfillment of proposed task. There were overall about 40 synapses in this system. The machine was built of tubes, motors, and clutches.

Content-addressable memory (CAM) is a special type of computer memory used in very-high-speed searching applications [**106**]. It is also known as associative memory, associative storage, or associative array. It compares input search data (tag) against a table of stored data, and returns the address of matching data (or in the case of associative memory, the matching data). Although not a full hardware system to support problems of machine intelligence, it is directly relevant to architectures that would support systems of full machine intelligence. High speed in search on hardware levels is one of the keys requirements to support real-time execution for intelligent systems. Several custom computers, like the Goodyear STARAN, were built to implement CAM.

Finally, IBM Watson [**55**, **59**, **67**, **128**], a research project and system that is capable of unstructured data analysis and natural language processing takes an important place in the cutting edge research performed in the area of AI and Machine Learning. Watson is also known in the AI community as a system that was able to outperform the former winners of the quiz show Jeopardy in 2011 [**67**], while utilizing 4 TB of storage and not being connected to the Internet.

2.5. Algorithms Performance Considerations

There was significantly less research efforts done to date in measuring performance of AI and ML, as compared to the overall amount of research effort in these fields. However, a few efforts present research on performance considerations in planning [**158**, **202**], expert problem-solving systems [**102**], generic machine learning [**37**], vision processing [**56**], facial recognition [**30**, **36**, **124**], including 3-D facial recognition [**32**], and parsing context-free languages [**15**]. Additional research endeavors include automatic speech recognition [**12**], dependency and semantic parsing [**27**], autonomous moving vehicles [**152**], lip reading [**138**], and handwriting recognition [**66**].

2.6. Summary

This chapter provides a detailed overview of prior research in the areas of most significance for machine intelligence. These areas include overview of advancements in AI and machine learning, including systems that supported symbolic computing in the past, e.g. Fifth Generation project, Macyma, LISP, and Prolog, and closed-loop system examples, including Winograd's SHRDLU, Wizenbaum's ELIZA and more recent approaches in dialogue systems.

Additionally, an overview of HPC systems and hardware systems that in the past enabled programs in the area of AI, including LISP Machines by Symbolics, CM-2, CSC-6600 and CAMs is provided.

Finally, prior attempts in performance considerations of systems of AI are summarized. Unfortunately, there has not been enough effort put into considerations of computational expenses and resource requirements for problems in this areas. This fact emphasizes the importance of research on resource requirements estimation for machine intelligence.

A number of concepts provided in this prior research review chapter support major principles of the model and architecture for machine intelligence, that are discussed in Chapters 4 and 5.

Next chapter provides a discussion of what generic Intelligence, Artificial, and Machine Intelligence are from various points of views. A definition that presents the requirements of a system for Machine Intelligence is introduced in the end of that chapter.

3

Intelligence and Machine Intelligence

This chapter provides a discussion of the most controversial problem of this research—definition of generic Intelligence, and subsequently Machine Intelligence (MI). A number of such definitions are summarized, varying by schools of thoughts, including philosophy, cognitive science, and Artificial Intelligence (AI). Although this research question has been given a thorough consideration by a great variety of disciplines, to date there is not a unified opinion on what Intelligence, AI, and MI are.

This chapter is organized as follows: a few definitions of generic Intelligence are provided and discussed; considerations regarding the concept of Machine Intelligence are then provided, and finally the term "Machine Intelligence" is introduced as a working definition for the purposes of this research.

3.1. Intelligence

A lot of efforts have been made in the past few decades in order to address the understanding, definition and mechanism that constitute Intelligence. Researchers in various fields of arts and sciences have provided hundreds of such definitions. A quote by R. J. Sternberg accurately defines this situation: "Viewed narrowly, there seem to be almost as many definitions of intelligence as there were experts asked to define it." **[76]**

However, a lot of such definitions share certain commonalities, often varying by a little degree of additional features. The generic definitions of intelligence, also referred to as "collective definitions" **[114, 115]**, are provided by groups of people or organizations, e.g. various dictionaries, encyclopedias, language databases, etc. Below are a few such generic definitions grouped by the

distinguishing feature sets that unite them. Note that some of these definitions belong to a few sets at the same time, e.g. they could both contain "knowledge" and "learning".

3.1.1. Knowledge/Memory/Facts/Skills. This group of definitions focuses on the intelligent entity's abilities to deal with knowledge, factual representation and any other relevant information. Intelligence is defined by various sources as:

1) "The ability to use memory, knowledge, experience, understanding, reasoning, imagination and judgement in order to solve problems and adapt to new situations." [**4**]
2) "The capacity to acquire and apply knowledge." [**140**]
3) "The ability to learn facts and skills and apply them, especially when this ability is highly developed." [**174**]
4) "The ability to acquire and apply knowledge and skills." [**173**]

3.1.2. Learning/Reasoning/Understanding/Thinking/Planning. This group provides definitions of Intelligence based on the ability to learn new information, understand and process such information directed at entity and think about it. Intelligence is defined by various sources as:

1) "The ability to learn, understand and make judgments or have opinions that are based on reason." [**91**]
2) "The ability to learn, understand, and think about things." [**121**]
3) "The general mental ability involved in calculating, reasoning, perceiving relationships and analogies, learning quickly, storing and retrieving information, using language fluently, classifying, generalizing, and adjusting to new situations." [**80**]
4) "Capacity for learning, reasoning, understanding, and similar forms of mental activity; aptitude in grasping truths, relationships, facts, meanings, etc." [**93**]
5) "Individuals differ from one another in their ability to understand complex ideas, to adapt effectively to the environment, to learn from experience, to engage in various forms of reasoning, to overcome obstacles by taking thought." [**145**]
6) "The ability to learn or understand or to deal with new or trying situations. . . The skilled use of reason." [**132**]

7) "A property of mind that encompasses many related mental abilities, such as the capacities to reason, plan, solve problems, think abstractly, comprehend ideas and language, and learn." [**193**]
8) "Capacity of mind, especially to understand principles, truths, facts or meanings, acquire knowledge, and apply it to practice; the ability to learn and comprehend." [**194**]
9) "The ability to learn and understand or to deal with problems." -Word Central Student Dictionary, 2006
10) "The ability to comprehend; to understand and profit from experience." [**133**]
11) "The capacity to learn, reason, and understand." [**199**]
12) "A very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience." [**72**]
13) "...The ability of an organism to solve new problems." [**16**]

3.1.3. Environment. Finally, this group defines intelligence from the point of view of interacting with environment that entity is subjected to. According to some sources, Intelligence is:

1) "...The ability to adapt to the environment." [**200**]
2) "...The ability to adapt effectively to the environment, either by making a change in oneself or by changing the environment or finding a new one." [**25**]
3) "...The ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria (as tests)." [**132**]

3.1.4. Summary. Summarizing the three groups of definitions provided above, the conclusion is that Intelligence is an ability or capacity of an entity possessing it to obtain information directed at it, process that information (by means of understanding, thinking, reasoning), store, memorize and be able to use it (learning, experience) quickly in an environment that the entity is subjected to. An important detail that is only mentioned in one of the references is "a capability to learn *quickly*" (see item 12 in Section 3.1.2). Although this detail is not paid much attention to among the provided sources of definitions of Intelligence, it happens to be very important, as will be discovered later in this chapter (see definition of Machine Intelligence in Section 3.3).

3.2. Artificial Intelligence

In addition to attempts in defining Intelligence in its broad sense, there has been multiple approaches to providing definition of Artificial/Machine Intelligence. AI has been considered by a myriad of researchers, ranging from early philosophers to formal mathematical definitions in logics, from Turing Test in AI to algorithms that play chess and even mimic human brain behavior or attempt to artificially rebuild certain processes of the brain. While projects, targeted at simulating the behavior of a human brain, open up interesting insights onto humankind's evolution and help to formulate new scientific problems, they do not address intelligence per se. A human is much more complex than simply an intelligent entity, with so many features that do not necessarily contribute to intelligence (instincts, emotions, etc.). A large number of research projects targeted at simulating human brain from neurobiology perspective, or taking the brain as a model for certain algorithms in Machine Learning, such as neural networks, although have certain successful applications, by no means deliver Artificially Intelligent systems.

While common sources, such as dictionaries and encyclopedias define AI as i) "the capability of a machine to imitate intelligent human behavior" [**132**], ii) "the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings" [**25**], and iii) the capability of computers or programs to operate in ways believed to mimic human thought processes [**40**], a more thorough way to provide the specific characteristics inherent to AI within its definition is required.

Legg et al. [**115**] provide an overview of a vast variety of definitions for both broad Intelligence and Artificial Intelligence by generic communities, as well as fields of philosophy, psychology, computer science and AI. As authors mention themselves, "...definitions [of intelligence] presented below are, to the best of our knowledge, the largest and most well referenced collection there is." This section lists definitions from various research efforts in the area. According to these sources, AI is:

1) "The ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral subgoals that support the system's ultimate goal." [**3**]
2) "Intelligence is the ability to use optimally limited resources – including time – to achieve goals." [**109**]

3) "Intelligence is the power to rapidly find an adequate solution in what appears a priori (to observers) to be an immense search space." [**117**]
4) "Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines." [**126**]
5) "[An intelligent agent does what] is appropriate for its circumstances and its goal, it is flexible to changing environments and changing goals, it learns from experience, and it makes appropriate choices given perceptual limitations and finite computation." [**157**]
6) "Achieving complex goals in complex environments." [**68**]
7) "Intelligence is the ability to process information properly in a complex environment. The criteria of properness are not predefined and hence not available beforehand. They are acquired as a result of the information processing." [**144**]

In conclusion of their work, Legg et al. [**115**] provide a definition of Intelligence as a summary of common features present in a vast number of definitions of Intelligence and Artificial Intelligence. From their point of view, Intelligence is:

1) A property that an individual agent has as it interacts with its environment or environments;
2) Related to the agent's ability to succeed or profit with respect to some goal or objective;
3) Depends on how able the agent is to adapt to different objectives and environments.

Combining these features and properties into one concise statement yields the following definition by the authors: "Intelligence measures an agent's ability to achieve goals in a wide range of environments."

3.2.1. Summary. An overview of various points of view to what constitutes Artificial Intelligence was delivered in this Section. A lot of these definitions express belief that Artificial Intelligence is simply an imitation of human intelligent behavior. Such definition is not feasible for purposes of this work, as it is not clear what constitutes human intelligence. Recent research shows, for example, that machines could beat humans in certain areas of IQ tests [**190**]. Does that imply that a full and true general AI has been delivered? Are humans "less intelligent" than the machines, performing deep learning algorithms? The answer to both questions is of course "no".

As one can observe, there is no unified opinion on definition of Artificial Intelligence, and even the definition, provided in previous Section, that is meant to summarize all prior efforts appears to be incomplete and missing some crucial details. Although the majority of definitions provided in this Chapter are not wrong, they do not fully reflect the components necessary to perform this research. In order to be able to accomplish the goal of this research, a definition that would satisfy all the requirements of the work proposed is needed. Next Section is devoted to precisely that—the definition of Machine Intelligence.

3.3. Machine Intelligence—working definition

As mentioned above, there is not a unified opinion in the community as to what AI is. Furthermore, definitions provided often lack imperative components of Intelligent systems. Finally, according to some of these definitions, the case should be that AI has been delivered.

For these reasons and to make it clear what space this research is situated in, a definition of Machine Intelligence is introduced. **Machine Intelligence** is a finite algorithm that enables an agent possessing it to:

1) Dynamically model and interact with external environment, including entities in such environment;
2) Satisfy a set of objectives and axioms and interact with external entities in accordance with such constraints;
3) Be self-aware by including a model of itself in the representation of the environment where it is located (physically and logically);
4) Be able to process the tasks in real time.

This working definition will serve as a foundation for building an abstraction for Machine Intelligent system and will allow to avoid many questions irrelevant to this project that arise in the area of AI. Therefore, it is important to understand the distinction between Artificial and Machine Intelligence when reading this thesis and to understand what a Machine Intelligent system implies. A discussion that clarifies the four elements comprising a system for Machine Intelligence follows.

A finite algorithm implies that it is a non-modifiable algorithm that is not subject to dynamic adaptive behavior. This only refers to the core of the Machine Intelligent system's algorithm,

which is supposed to stay non-modifiable over substantial amount of time (revisions performed by system's engineers do not count, however the system itself should not be able to modify its low-level functionality). This matter is discussed in much greater details in Chapters 4 and 5.

An agent able to dynamically model and interact with the external environment means that there is an internal representation that is constantly changing and being updated as well as refined in the presence of external stimulus. To dynamically interact means that in addition to accepting input stimulus, the system provides some or multiple forms of output that affects its external environment. Such interaction can be a form of communication to provide information, it can be a manipulation of the immediate context such as moving an object in physical space, or it can adjust its own location.

Real time requires that the rate of operation, including modeling of context and interactive response, be consistent with the time constants of the external environment and included entities with which it is interacting. This means that it must be fast enough to support meaningful process, of which it is a part, but time-aware such that it does not try to do something at too fast a rate such as flying an airplane, delivering a speech, or presenting a movie to a group of external entities.

The external environment will include the physical neighborhood (line of sight and hearing) of the intelligent system permitting interaction media including direct connect, audio, visual, and tactile communication. But it can also involve logical neighborhood, with entities of which the system is interacting via certain means of communication, including telephone lines, networks, radio communication, or remote sensors data.

Individual intelligent entities are other systems within the local or logical neighborhood of the intelligent system that themselves satisfy the definition of intelligence, such as people or other systems of MI with which the system in question is interacting.

The concept of a set of objectives, functions, and axioms is of great importance. It identifies the most complicated part of the Machine Intelligence. In addition to a hierarchical system of dynamically evolving and overriding objectives, accompanied by contexts associated with them in order to enable situational awareness, a set of pre-defined non-mutable axioms is imperative to any such system. Without such axioms, there is no way to estimate what actions the system might consider optimal in solving a particular objective. They exist in order to establish a rigid set of constraints that would proactively eliminate any extreme possibilities under any circumstances.

Another important consideration for MI are notions of expectation and common sense. While the former is discussed in Chapter 4, the latter has been a focus of the recent research activities by, among others, the Allen Institute for AI [**181**, **182**, **201**].

3.4. Summary

This Chapter serves a dual purpose—on the one hand, it attempts to overview and provide definitions of what is considered to be Intelligence and AI by scientific and broader general community; on the other hand, an argument stating that such definitions, although not incorrect, often lack some crucial details in defining AI. In order to situate the present research into a necessary problem space, a definition of Machine Intelligence that is considered to fully explain all aspects of intelligent system, is introduced. It is very important to understand that this definition is not meant to compete with those mentioned above in this chapter, nor is it clear whether it provides a complete definition of general AI. However, it describes to the full level of detail aspects of Machine Intelligence that are required to conduct this research effort.

Next Chapter introduces an abstract model for Machine Intelligence, called the Abstract Architecture.

4

Abstract Architecture for Machine Intelligence

This chapter presents the most integral piece of this thesis—a definition of a substantial part of the model for Machine Intelligence (MI), called the Abstract Architecture. Together with the Workflow for MI, described in Chapter 5, Abstract Architecture defines full model of MI. The architecture definition is provided in its entirety, therefore the present model can be considered a fully defined abstraction for a system that is capable to obtain, process and react upon stimuli stream it is subjected to.

Prior research review provided in Chapter 2 shows a few approaches where conceptually similar, but less detailed architectures were used in order to provide definitions for intelligent systems. Among them are very specific applications for mobile robot navigation [**151**], as well as approaches to defining system architectures in the area of general Artificial Intelligence (AI) [**110**, **111**, **122**, **154**].

This chapter is structured as follows: Section 4.1 provides a top level overview of the abstract architecture and discusses considerations of system requirements of the architecture's modules on this level. Section 4.2 provides overview of knowledge state representation and the role it plays in the entire model of abstract architecture. Next, Section 4.3 discusses available symbolic methods—the mechanisms that are used to enable the MI system's understanding and learning capabilities. Section 4.4 provides a brief overview of the front end requirements and considerations for the system, followed by the chapter's summary in Section 4.5.

The distribution of components of the abstract architecture is categorized in a hierarchical manner. Thanks to such representation, detailed design of the abstract software hierarchy for MI is provided. Exhaustive definition of all the components, and further the interaction between these components via the workflow, provides a full and complete model for MI.

4.1. Abstract Architecture Overview

The abstract architecture that models Cognitive Real-time Interactive System (CRIS) is comprised of interrelated autonomous components, each of which serves respective key functions in the entire system operation [**177**]. These components can be categorized into three groups at the top level, and are as follows (see Figure 4.1):

- **Knowledge state** represents various types of knowledge that the system is continuously maintaining throughout its execution;
- **Symbolic methods** include means for processing the accumulated knowledge as well as any other symbolic-based data;
- **Front-end** provides means for the system to exchange information with the outside world.

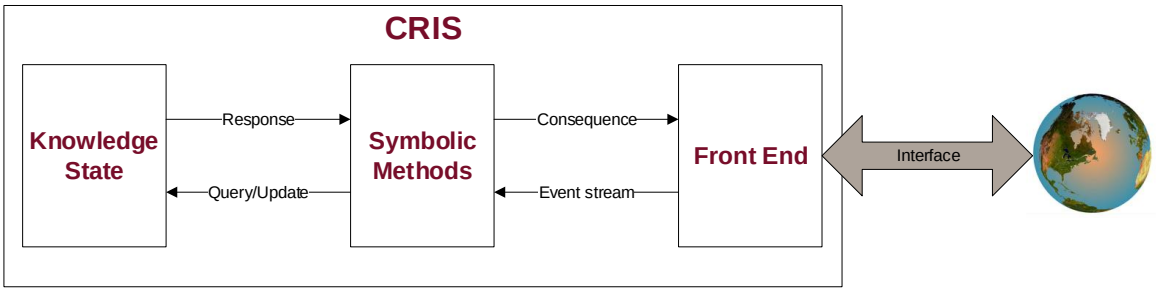

FIGURE 4.1. Abstract architecture of CRIS

Such a system represents a closed execution loop by obtaining data stream at its front-end, which is then processed to obtain decisions about updating the knowledge state. This updated state is subsequently used to provide an output stream to the external environment. The external world modeling, performed by various research efforts in the past, is an important consideration when dealing with intelligent systems and robots. It is not considered in this thesis, however, given that existing methodologies on formulating the world model exist [**46**, **151**].

Another inherent aspect of a CRIS is the notion of real-time. It is imperative for the system to operate in a real-time environment, delivering results within the expected timeframes and formulating objectives and further sub-tasks accordingly [**64**]. Therefore, time is intrinsically included in the system's knowledge state.

Knowledge state defines the state of a system at any given point of time, as well as contains means for evaluation of its environment, surroundings and internal state. The internal mechanisms of knowledge state allow for the system to constantly keep updated track of its identity, physical location in space, location in time and operational status. Additionally, this element allows the system to maintain information about various objectives that need to be achieved, as well as situational contexts, associated with them.

Symbolic methods represent a collection of approaches that define how data is processed. These approaches include learning, update mechanisms, planning, inference and conflict resolution. All of these modules are described in detail in Section 4.3, while the question of how and when they are used during CRIS operation is considered in Chapter 5. Further, the resource requirements are modeled by a performance model, which is based on the cost analysis of the abstract architecture components and their incident rates, defined by the execution workflow.

Finally, the front end contains a number of interfaces that the system uses to communicate with the external environment. These include speech, visual and interconnect devices, as well as sensors that collect audio, video and physical I/O for the system.

4.2. Knowledge State

Knowledge is an essential part of any system of MI which requires a well-defined representational hierarchy. A wide variety of knowledge representation techniques exist, including knowledge databases, ontology representation, and production systems, among others [**23**,**24**,**42**,**49**,**147**].

The proposed architecture differentiates knowledge into a number of classes varying from relatively static foundational knowledge to most rapidly changing imperatives derived from input stream commands or otherwise implied by local context.

Another way knowledge is differentiated is by braking it into universal knowledge, that any CRIS instance might possess, such as fundamental facts about geography, history or physics, and unique knowledge (Figure 4.2), that any system belonging to a particular environment needs to maintain, such as the machine's location, its internal system status, or what other agents it is currently interacting with in its surroundings.

4.2.1. Universal knowledge. This knowledge type is a collection of facts which represent that knowledge, drivers that organize, structure, process and obtain new knowledge and, finally, the

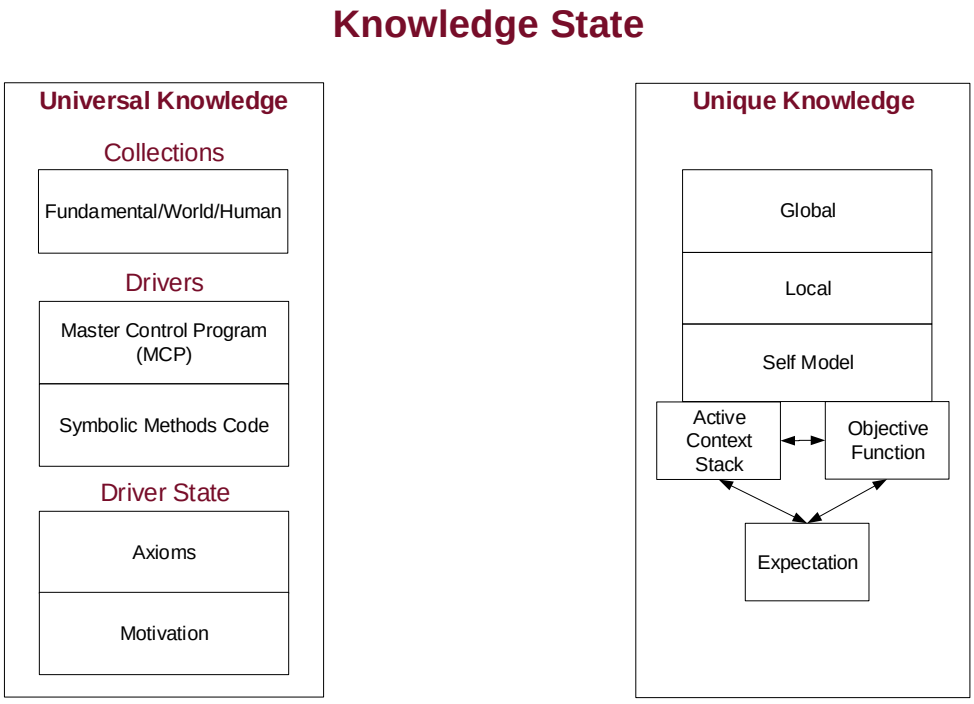

FIGURE 4.2. Knowledge State Representation: Universal and Unique Knowledge

driver state, which allows the system to behave in accordance with expectations that are implied on the system. It is important to distinguish universal knowledge (Figure 4.2), provided to any instances of CRIS, and, possibly, universally maintained throughout all instances in a shared knowledge repository, versus the unique knowledge, which is accumulated and maintained as a result of the particular system instance's personal experiences and interactions.

4.2.2. Types of Universal Knowledge: Fundamental, World and Human. These types of knowledge represent facts about corresponding domains of knowledge type. The Fundamental knowledge represents laws of the Universe that hardly ever change. Examples of such knowledge types include laws of mathematics, physics or chemistry. The World knowledge contains all the information regarding the world without humans, while Human knowledge module includes knowledge of humanity, including human history, art, philosophy, etc. These types of knowledge share relatively common data structure representation, which includes the object identifier, structured representation, raw representation, type, source and confidence level. The structured record and type are inferred by the system with a certain confidence level from the corresponding raw

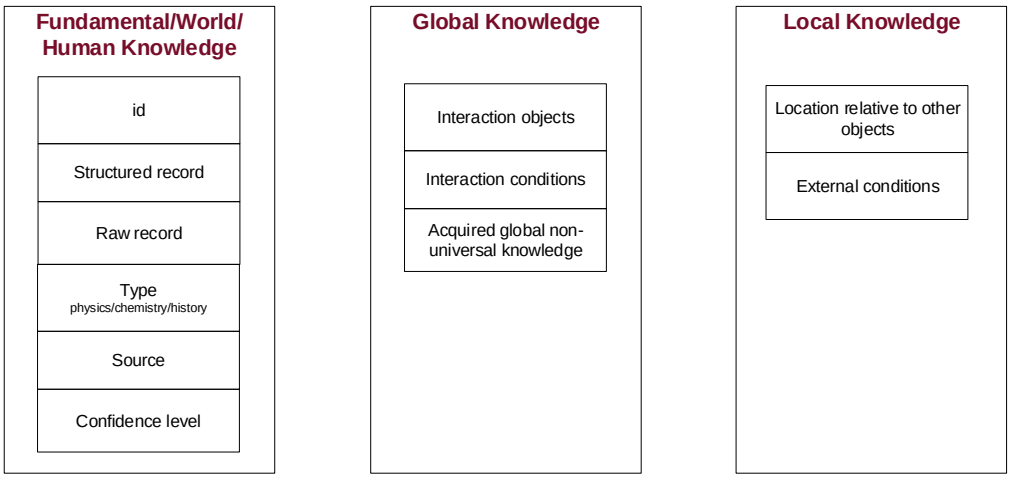

FIGURE 4.3. Types of knowledge: global, local, and knowledge collections

record, which is the original representation of any stimulus the system gets from one of its inputs. It is necessary to keep track of raw records, as sources and the system might re-evaluate the previously obtained structured record representation due to some newly acquired or internally updated information.

4.2.3. Master Control Program. A mechanism that drives the knowledge query and update, as well as most of the other critical actions the system undertakes, is referred to as Master Control Program (MCP). The term was initially introduced by Burroughs corporation, where MCP was a proprietary operating system on one of the early machines, Burroughs B5000 [**28**]. The term later became popularized in a Sci-Fi movie "Tron", where MCP was a computer program that evolved itself and decided to get rid of the humans.

The MCP in CRIS is analogous to central nervous system for autonomic behavior. It ensures that a system is constantly going through an outer loop, executing required actions in bounded time. During the execution of a loop cycle, the MCP needs to perform a number of actions, including:

1) Query the system's status by interacting with the self model[1] module;
2) Satisfy the objectives that are due this execution cycle;
3) Query all the I/O sensors[2] to obtain updated information about the external environment;
4) Trigger the knowledge state update mechanisms, and

[1]definition provided further below in this document
[2]See Sec. 4.4

5) Update current Active Objective Function Stack and Active Context Stack hierarchies.

When satisfying these objectives, MCP needs to obey the axioms (see Section 4.2.5 below) that are imposed on a system and are hard-coded into the system's non-volatile read-only memory. This step is necessary in order to guarantee that the system will not become dangerous to the world and human beings under any circumstances. These rules can not be altered by the system itself and would require external interference (if need be). During the cycle of the execution loop, a number of symbolic methods are executed, depending on the specific task, which are described below in Section 4.3.

4.2.4. Symbolic Methods Code. References to the symbolic methods are present in two different places in the abstract architecture—as a separate entity (see Figure 4.1), as well as as a part of the universal knowledge module in Figure 4.2. The difference between these two modules is that the symbolic methods in Figure 4.1 represent the actual code instances (executables) that perform the tasks assigned to them, while the symbolic methods code in Figure 4.2, as an element of universal knowledge, represents, in fact, **the knowledge** about these methods, and, more importantly, CRIS is capable of updating these methods, as any other knowledge elements, if necessary. As the newly updated instances of symbolic methods become available, it is the MCP's task with the help of update utilities to introduce new instances of these methods for operation.

4.2.5. Axioms. The axioms is an integral component of CRIS architecture, which has to be present in the system's workflow (discussed in detail in Chapter 5). This step ensures that the predefined axioms that reside in non-volatile read-only memory do not get violated by performing the action currently processed. Every loop iteration in CRIS's processing the MCP needs to obey the axioms that are imposed by the developers and are hard-coded into the system's non-volatile read-only memory. This approach is necessary in order to guarantee that the system will not become dangerous to human beings (or any other entities that need to be protected) under any circumstances. These rules can not be altered by the system itself and would require external interference (if need be). The check is performed on two levels to insure no harm is intended by executing the desired objective—the abstract level and the concrete level. Should there be any, even a slight possibility, of any of these steps of the current objective bringing harm, the whole

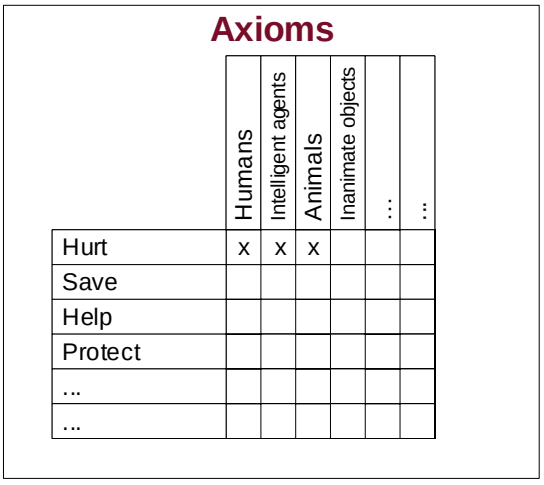

FIGURE 4.4. Axioms

objective is reconsidered in order to obtain a new sequence of actions, get some modifications from external world or abstain from performing the objective.

4.2.6. Motivation. Motivation, or Utility function, $U : s \rightarrow R$, a term accepted in the wider AI community, is a performance measure that allows for quantification of the system's internal states, s [**165**]. In other words, this function describes a degree of "happiness" or "satisfaction" that the system achieves by being in a certain state.

4.2.7. Unique knowledge. The unique knowledge, in addition to various facts about the environment and its elements, constantly maintains information about itself[3], about the goals that are subjected to it, as well as any contexts relevant to those goals.

In addition to containing global and local knowledge, the unique knowledge maintains information about the self model, Active Context Stack and Objective Function (OF) (see Figure 4.2). These three components working together present one of the key intelligent functionalities of the entire system. While self model is constantly monitoring and updating the "health" state and resource utilization of the system, objective function and Active Context Stack combined together present a foundation for formulating new tasks in relevant contexts.

4.2.8. Global and Local Unique Knowledge. The unique knowledge is once again broken into two separate classes, both pertaining to the experiences, uniquely acquired and maintained

[3]Hence, the system is self-aware, according to our definition, as in addition to keeping the model of surrounding environment, it can identify itself in the environment and in relation to other members

in a particular instance of CRIS. The global knowledge stores facts of agent's interaction with the world that are not in its physical or logical proximity; the local knowledge, on the other hand, contains facts that belong to events, occurring in the agent's immediate proximity. An example of global knowledge fact could be that a certain person—who the agent needs to keep track of—is presently located somewhere overseas, while having a conversation with that person on a phone line could be treated as a local knowledge fact during the conversation, as both the agent and the person would be located in the logical proximity at that time.

4.2.9. Self Model. Self model as a subcomponent of the unique knowledge partially represents the state of the system at a certain time instant. It contains information about the system's capabilities, "health" status, utilization status, progress towards goal, history of itself and spatial dynamics. All these components are data, i.e. objects that are not driving mechanisms themselves, but a collection of knowledge that evolves (gets created/modified/deleted) over time. Self model along with two other components, Active Context Stack and aclOF, represent the core functionality of CRIS. The interrelation of these 3 components is of great importance and is defined below in Chapter 5.

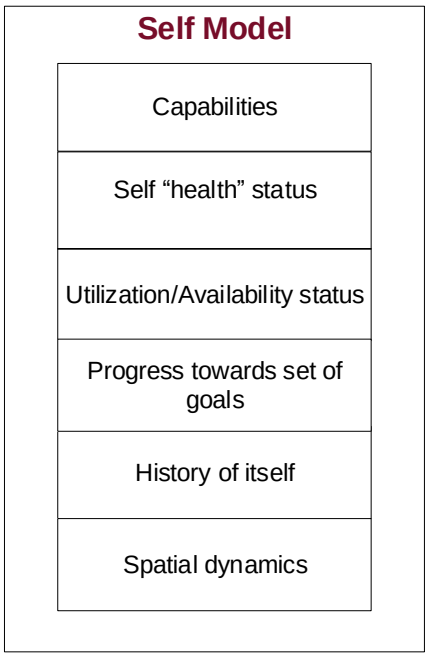

FIGURE 4.5. Self Model

4.2.10. Active Context Stack. Active Context Stack (ACS), presented in Figure 4.6 contains frames of previously saved contexts. The frames of ACS are tightly coupled with frames on Active Objective Function Stack (AOS). Therefore, ACS frames have to keep track of what AOS frames they are related to. In addition to AOS frame pointers, the ACS frame includes id and information about the context content, alongside with child, parent and sibling frames. In case the system inferred incorrect content information, a pointer to the raw input is provided (should the system need to roll back and re-evaluate the context in question). The ACS, along with the AOS, described below, are the most dynamically changing data structures, as the system is constantly processing and updating its knowledge state through these two components.

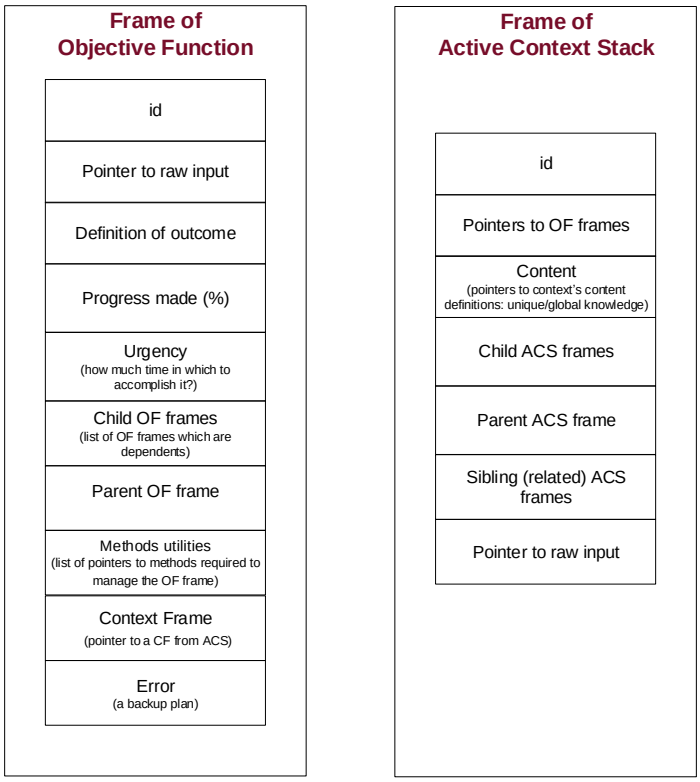

FIGURE 4.6. Frames of Objective Function Stack and Active Context Stack

4.2.11. Expectation. The expectation is another utility that plays a significant role in decision making when attempting to understand current objectives and contexts while processing input stimuli[4]. The expectation guides the decision making at different levels of detail when processing

[4]see Chapter 5 for explanation of how the stimuli are introduced and processed.

takes place. It helps to pick correct context, and, therefore, to make a decision in favor of a particular outcome, on the level of audio, video and textual processing in the pre-processing stage. It also becomes helpful when forming sub-tasks due to inability of the system to perform required objectives right away, when it needs to be broken down into imperative sequence of steps. The role of expectation is to considerably reduce the search space of possible objective functions by associating pre-existing contexts and prioritizing them.

4.2.12. Common Sense. Common sense is required for CRIS as supplementary mechanism that would allow to exhibit intelligent behavior. The common sense engine maintains the system knowledge state about the facts that are inferred by association with relevant contexts in an attempt to satisfy certain objectives. An example of a common sense behavior is that the system is aware that a man had a newspaper and was in the living room. The man later went into the kitchen, holding the newspaper. The common sense engine should therefore infer that the last known newspaper location is the kitchen. The topic of common sense is a separate area of research in the AI community with a wide variety of applications ranging from business systems to vision processing and natural language understanding [**143**].

4.2.13. Objective Function. The OF specifies in great details the objective that is to be achieved by the system (see Figure 4.7). In order to provide a full problem specification, the OF contains information about goals, requirements and jobs currently being performed. Alongside with this information, there is a curiosity factor that is considered. AOS is defined as a part of the OF. The frame of AOS is presented in Figure 4.6 and contains in addition to the fields similar to ACS, such as id, pointers to raw input and pointers to related child and parent OF frames, the definition of outcome, progress made, symbolic methods utilities that are required to achieve the desired objective as well as pointers to related contexts.

4.3. Symbolic Methods

CRIS's symbolic methods include learning, update mechanisms, planning, inference and conflict resolution. All of these methods are independent actors, that are operating on required tasks, taking current context information into consideration. Learning is one of the most essential and important methods allowing the system to modify its future actions based on the empirical information about its interaction with the environment. It, possibly, is the only actor that can modify

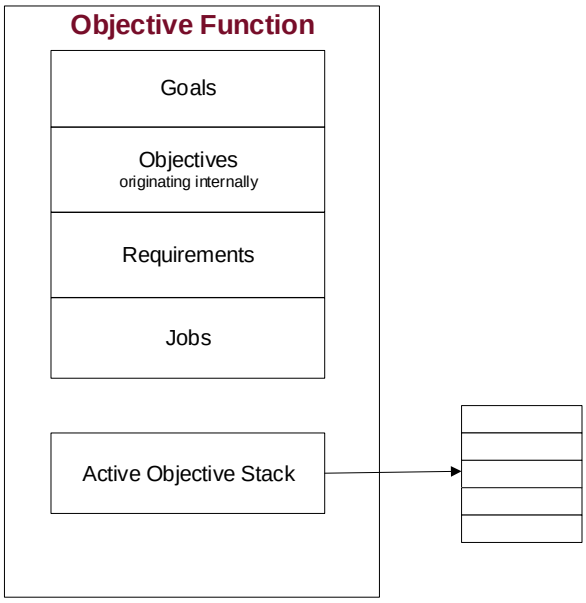

FIGURE 4.7. Objective Function

the behavior of other actors, including, itself. The update mechanisms ensure that the new information is processed and recorded in the machine's knowledge state base. Planning module supports for a plan derivation for various tasks. In case a machine does not know what type of metrics to use, or has to achieve a goal while generating a plan, it is attempting to utilize learning modules in order to express the goals through currently existing knowledge or obtain new knowledge about the problem. Finally, inference mechanisms allow to draw new conclusions about the facts available in the knowledge state, while conflict resolution ensures that all the constraints of present goals are satisfied.

4.3.1. Learning. Learning is one critically important component of any Machine Intelligent system that has not been successfully addressed to date. The advancements in areas of AI and Machine Learning suggest certain intelligent-like behavior for certain scenarios, however recent research endeavors have been facing complications in delivering domain-independent generic intelligent agents, capable of learning like humans do. The complication arises due to inability of ML approaches to approximate truly complex functions [**13**]. Research in domain-specific areas, where agents learn certain requirements of interacting in the environment, include reinforcement

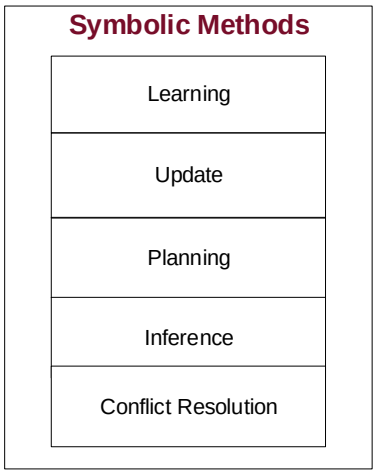

FIGURE 4.8. Symbolic Methods

learning, neural networks and genetic algorithms, Bayesian learning, etc. [**165**]. CRIS's learning mechanism is designed such that it should be able to:

a) obtain new knowledge from external sources and update the internal knowledge base appropriately,
b) modify its behavior (i.e. learn and be able to change its *code*, including the MCP, if necessary, and any of the symbolic methods utilities, as well as introduce new such utilities, if needed).

Systems of MI will have to posses this behavior in the future.

4.3.2. Update. The update mechanisms provide a way for the system to perform updates to the corresponding data structures and knowledge management module it maintains. In order to better understand how the update mechanisms are triggered, consider a workflow diagram in Figure 5.2 of Chapter 5. Update mechanisms are initiated multiple times throughout one CRIS's workflow execution loop. First, an update occurs when work on existing context, rather than the new context, is being performed. This update to the system is immediately followed by updates to the self-model. Additionally, an update to objective function is performed, once objective is identified from stimulus.

4.3.3. Planning. There has been a considerable development in the field of planning in the past years, with increased performance and scalability. That was possible due to the assumption

```
(define (domain gripper-strips)
    (:predicates
    (room ?r)
        (ball ?b)
        (gripper ?g)
        (at-robby ?r)
        (at ?b ?r)
        (free ?g)
        (carry ?o ?g))
    (:action move
        :parameters (?from ?to)
        :precondition (and (room ?from)
        (room ?to) (at-robby ?from))
        :effect (and (at-robby ?to)
        (not (at-robby ?from))))
```

FIGURE 4.9. PDDL domain

```
(define (problem strips-gripper2)
(:domain gripper-strips)
(:objects rooma roomb ball1 ball2
left right)
(:init (room rooma)
(room roomb)
(ball ball1)
(ball ball2)
(gripper left)
(gripper right)
(at-robby rooma)
(free left)
(free right)
(at ball1 rooma)
(at ball2 rooma))
(:goal (at ball1 roomb)))
```

FIGURE 4.10. Sample problem

a lot of this planners require regarding the space they operate in, which is normally assumed to be known in advance [**44**, **161**]. Virtually all of the state of the art planners operate with Planning Domain Definition Language (PDDL), which allows to identify an abstraction of the world, its objects and interrelationships between those objects (predicates). Typical components of a PDDL planning tasks include: objects, predicates, initial state, goal specification and operators. A typical planner expects two input parameters, a domain file that describes the predicates and actions, and a problem file that provides specifications for objects, initial state and desired goal state.

A typical small example PDDL domain and problem are presented in Figure 4.9 and Figure 4.10. The `gripper-strips` domain has a number of predicates, such as `room`, `ball`, `gripper`, etc., as well as one action called `move` defined. The problem named `strips-gripper2` identifies what objects (names) are present in the domain, initializes them using the predicates previously defined and sets a desired goal to be achieved.

A typical planner architecture [**86**] looks like the one presented in Figure 4.11. The planner solves a task in three phases:

a. The Translation component is responsible for transforming the PDDL input into a planner's internal representation. During that process, a number of normalizations occur, in order to bring the logic formulas to a format, understandable by a planner. Most importantly, an invariant synthesis method is used in order to identify groups of related

propositions, which can be encoded as a single multi-valued variable. The output of the translation component is a multi-valued planning task (MPT).

b. The Knowledge Compilation generates four kinds of data structures that are key for search step. Domain transition graphs encode how and under what conditions the state variables change their values. The causal graph represents hierarchical dependencies between different variables. The successor generator is a data structure for determining the set of applicable operators in a given state, while the axiom evaluator is a data structure for computing the values of derived variables.

c. The Search component implements different search algorithms to perform the actual planning. Two of these algorithms, greedy best-first search and multi-heuristic best-first search, utilize a heuristic function in order to perform this search, while focused iterative-broadening search uses information encoded in causal graphs.

For more detailed description of this architecture and its components, refer to [**86**].

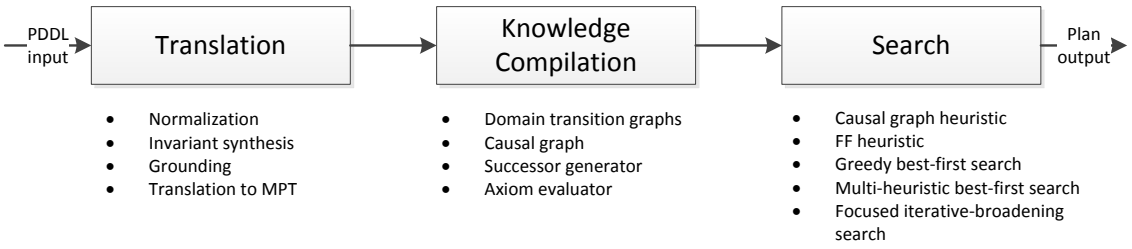

FIGURE 4.11. A Typical Planner Architecture

4.3.4. Inference. Inference mechanism play an important role in aiding commonsense module and maintaining updated knowledge base by supplementing it with the newly inferred facts. Systems utilizing automatic logical inference were a popular research topic in the past with applications in expert systems and business rule engines. More recent work was done using automated theorem proving engines with the help of formal logic. The most well-known algorithms that use first-order logic for inference are forward-chaining, backward-chaining and resolution.

4.3.5. Conflict Resolution. The conflict resolution method insures that a right decision is picked during the decision-making process that would optimally satisfy the motivation (utility

function) and the objective function (see Sec. 4.2.13). This utility, essentially, performs an optimization, choosing the most feasible solution across all the relevant parameters. In case there are several potential plans (and schedules) that the stimulus processing has come up with, the most feasible outcome is determined by the conflict resolution method.

4.3.6. Progress Towards Goal. Progress towards goal serves as a success measure and how fast the system is converging towards the set goal. Given with the real-time requirement of agents in certain applications, this measure becomes really important, as it allows the system estimate the rate at which certain milestones are to be achieved.

4.4. Front End

The Front End includes a variety of I/O interfaces that CRIS uses in order to communicate with external world. These include speech, visual, interconnect, along with physical and tactile sensors. Some possible examples of such sensors are ultrasonic, infrared, and laser range distance sensors, microphones (speech), cameras (visual), sound, temperature and tactile sensors.

The goal of this research is not to explore the I/O interfaces per se, therefore they are provided in this abstract architecture for the sake of completeness. Existing extensive research results in robotics, human-computer interaction and other related areas provide abilities to utilize existing software solutions. Some of these solutions, however, themselves can require considerable amount of computational resources, but existing results for such systems will be used in order to make the estimates.

4.5. Summary

This chapter introduces one of the integral and most important pieces critical to this research—a part of the model for MI named Abstract Architecture. Together with the Workflow for MI, which defines interrelations between components of the Abstract Architecture and is described in Chapter 5, Abstract Architecture defines full model of MI. The architecture definition is provided in its entirety. Therefore the present model can be considered a fully deterministic abstraction for a system that is capable to obtain, process and react upon stimuli stream that it is subjected to.

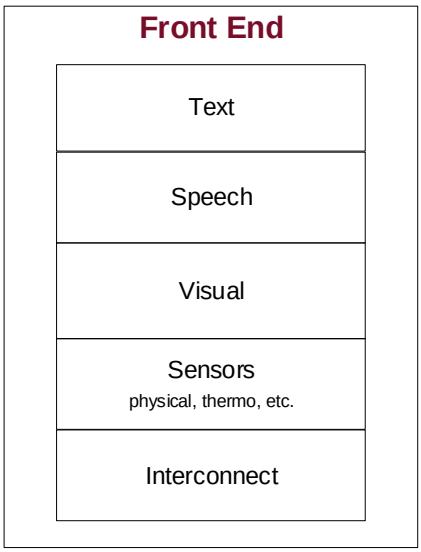

FIGURE 4.12. Front End

The discussion provides detailed representation of the architecture, beginning with a top-level view for the system that incorporates three components—Knowledge State, Symbolic Methods, and Front End.

One of the key concepts of the Abstract Architecture is an ability to process external stimuli by forming objectives that are associated with the environment's contexts, aided by common sense and expectation modules.

Some of the symbolic methods are discussed in detail in this chapter, including planning, update and inference. A variety of Front End mechanisms required to support intelligent systems is provided.

Now that the introduction to the model of MI is made in this chapter, the next chapter will discuss the second part of the model, namely the Workflow.

5

Workflow for Machine Intelligence

While the abstract architecture described in Chapter 4 provides an overview of CRIS, it does not focus on how the components of abstract architecture interact within the system. Moreover, it is not apparent how this complex entity takes on making decisions, when attempting to respond to external commands or requests, which are generalized into a term *stimuli*, as well as how the knowledge is stored internally and what mechanisms are triggered during such decision making processes. Furthermore, estimating CRIS's resource requirements by only considering the architecture would not be possible, as it is not clear at what incident rates the functional elements of the abstract architecture operate. The exploration of incident rates is performed in greater detail in Chapter 6, which introduces and discusses the performance model and resource metrics for Machine Intelligence. The connection of the workflow and performance model allows to provide the resource requirements estimates. CRIS workflow is therefore introduced in an attempt to get closer to answering the main question of this work, which is to obtain estimates of the lower bound resource requirements for Machine Intelligence.

The workflow description provided below serves an important role of introducing the internal operational behavior patterns of a typical Machine Intelligent system. Detailed descriptions of internal mechanisms that are involved in processing of any signal that is captured by the system are provided. In addition, it is shown how the system deals with solving complex objectives. This chapter also discusses how any Machine Intelligent system deals with potential infinite loops when subjected to not being able to make a decision. This chapter, alongside with Chapter 4, is of significant importance to understanding the internal mechanisms and overall organization of the model for Machine Intelligence.

5.1. Simplified Workflow Model

A simplified workflow diagram is presented on Figure 5.1 and describes a typical iteration of a Machine Intelligent system. Boxes indicate elements of abstract architecture. Straight arrows show the flow of execution, while dashed arrows signify dependency of components on Knowledge State.

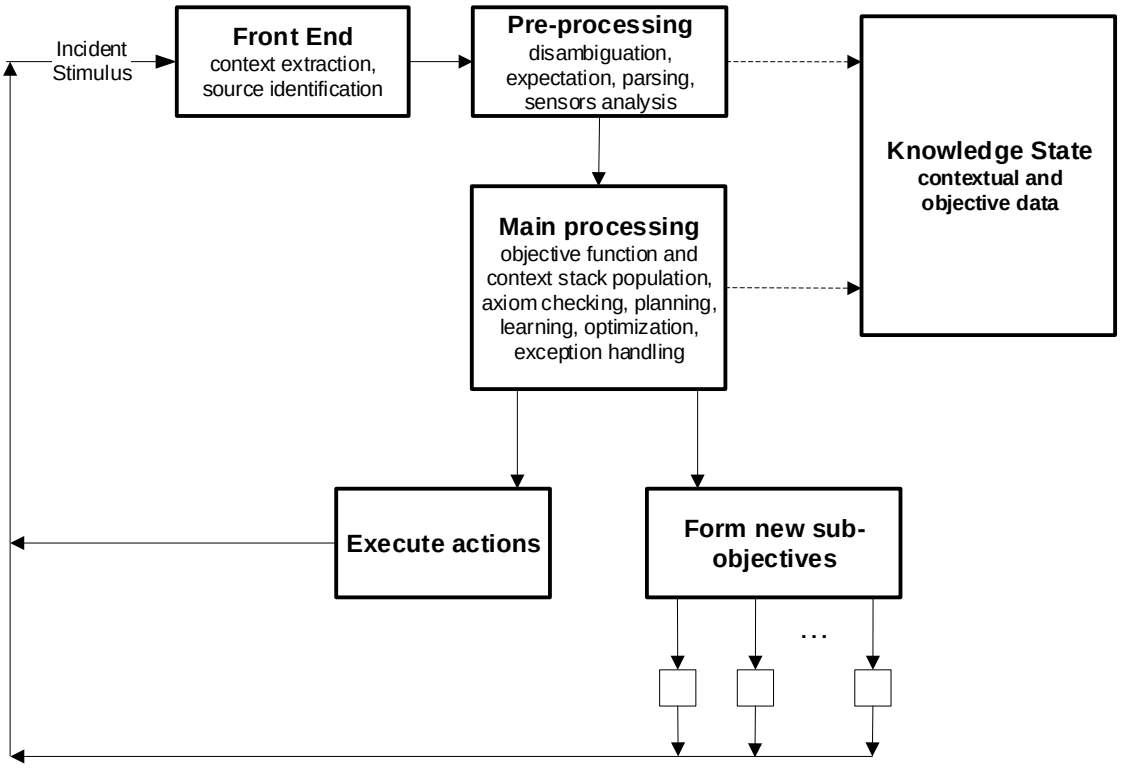

FIGURE 5.1. CRIS basic workflow

Any iteration begins by processing an incident stimulus that arrives to CRIS's front end and is stored in a priority queue, awaiting to be processed. Such queue is required for CRIS to address stimuli and subsequent potential tasks and objectives according to the level of urgency that is associated with each stimulus. In short, pre-processing attempts to identify a type of stimulus, e.g. text, speech, video, etc., being processed, estimate the context associated with stimulus (see full workflow description below in Section 5.2 for details), acquire and analyze sensors data, and evaluate importance of input stimuli.

After initial pre-processing takes place, the workflow execution proceeds to a thorough analysis of given stimulus, going through steps which are kept abstract in this basic workflow diagram. During this stage, a number of updates to the knowledge state occurs, facilitated by the symbolic methods and master control program (MCP). During this main processing stage it is decided whether the context in question existed previously or, alternatively, the system is dealing with a brand-new context.

Objective functions and relevant context stack frames are either updated, in case of pre-existing contexts, or newly populated. Further, objectives are identified from the stimulus being processed. Sanity axiom satisfiability checks are performed on various levels (low and high abstract level) in order to determine that there are not harmful actions scheduled to occur. Planning and learning mechanisms are then triggered in attempt to satisfy the objective function.

Finally, in case a feasible action scenario—or, potentially, several scenarios—have been devised, a declarative sequence of actions is produced and executed. Alternatively, CRIS might form new sub-objectives, that may become recursive dependencies to be satisfied in order to achieve the principal objective. Additionally, in case no feasible solution is found, an exception is raised in an attempt to resolve the issue of processing stimulus with the help of external guidance.

Exceptions might potentially occur during many stages of workflow execution. Again, detailed analysis of exception raising and handling is provided below in Section 5.2 on detailed workflow description.

Finally, the control goes back to the entry point of the workflow, taking on a new stimulus or proceeding to recursively process any sub-objectives that were formed and were placed in the priority queue.

Main processing on Figure 5.1 is expected to be the most resource demanding part of the basic workflow with pre-processing and front-end being less computationally expensive, yet still of significant importance in some applications of Machine Intelligence, regardless of the degree of consumed resources.

This basic representation of a workflow for Machine Intelligence, in spite of its present high-level representation, establishes a top-view representation that *any* Machine Intelligent system, at least for the purposes of this research effort, follows. Even though certain crucial technical details

are hidden in the description of basic workflow, the four most important points of a MI system's workflow to be emphasized are:

1) Closed-loop workflow execution model;
2) Recursive nature of goal/objective formulation and processing;
3) Context and objective function components closely interacting with each other throughout any single processing cycle;
4) Modular design of the workflow—which is inherited from abstract architecture representation—with three main components: front end, stimulus processing, and knowledge state.

5.2. Full Workflow for Machine Intelligence

Following is the full CRIS workflow presented on Figure 5.2. Boxes represent elements of the abstract architecture, straight arrows show the execution flow, while dashed arrows signify elements that are recursively dependent on other tasks. Colors are as follows: green indicates successful objective formulation and execution, red signifies an error or exception, while yellow stands for formulating a recursive task to be satisfied.

Processing of any request typically begins with the system's **front end** receiving input signals, which are called *incident stimuli*.

The functional representations of workflow's main processing elements is provided throughout this chapter in the following format:

```
function_name
input: parameter name(s)
output: parameter name(s)
```

Such functional representation provides clear understanding of the data flow that is occurring throughout the execution of CRIS execution cycle and identifies the functional elements alongside with their input and output parameters that are expected by the modules of CRIS model. In addition to the description of the workflow's abstract functional elements, two working examples that show how these elements are operating in a non-abstract way will be considered below in Sections 5.3 and 5.4 of this chapter respectively. These examples include a case of autonomous moving vehicle (AMV) [**63**, **79**, **183**, **187**] and T. Winograd's blocks world [**196**–**198**]. Following is the description of CRIS full workflow's functional elements.

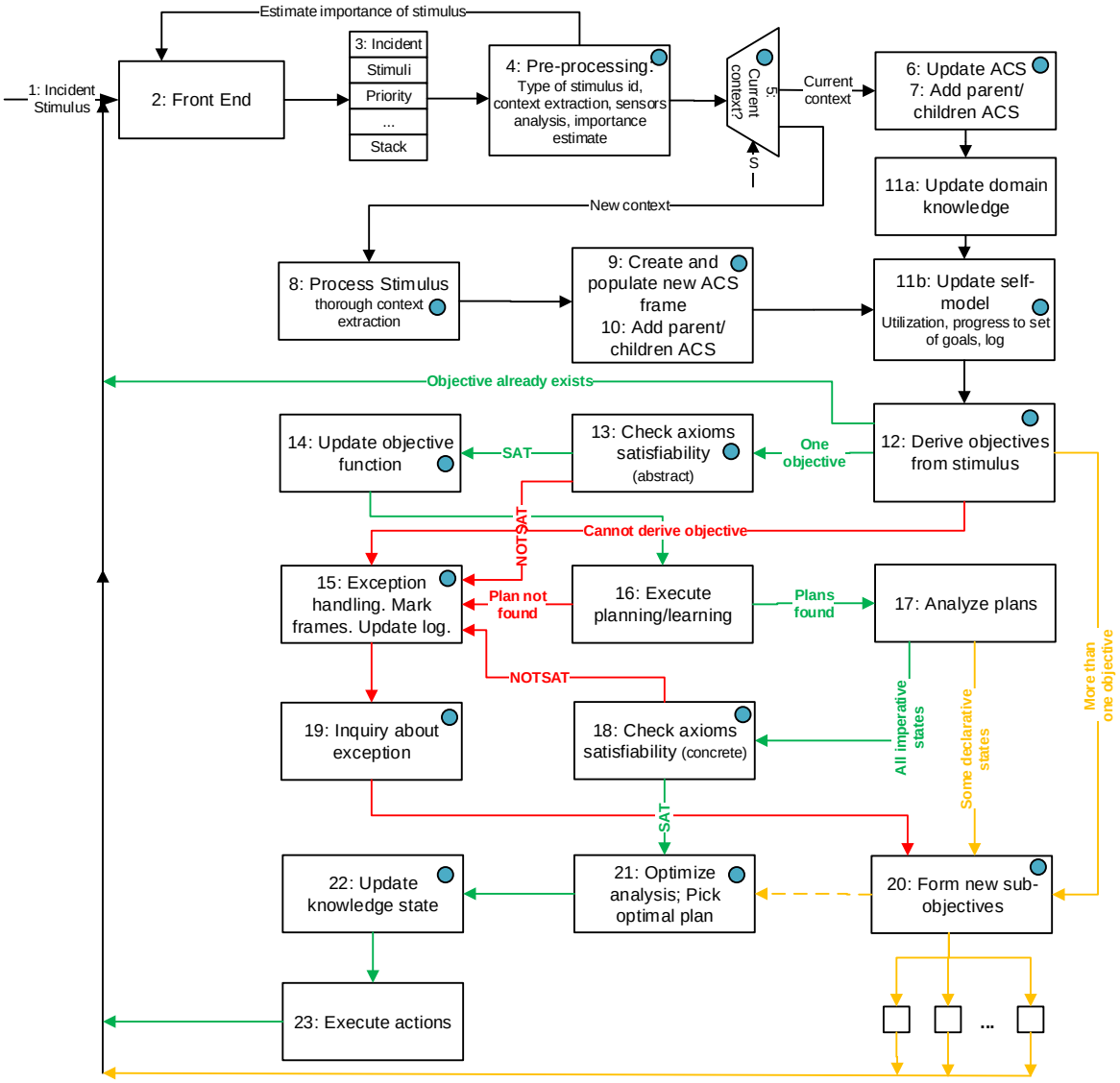

FIGURE 5.2. CRIS full workflow

5.2.1. Front End. Any stimulus is initially received by CRIS's front end module. The front end representation is as follows:

```
front_end
input: stimulus
output: stimulus internal representation
```

The front end element obtains stimulus information from the I/O devices and CRIS's sensors. The types of input stimuli data vary among text, speech, visual (including photo and video), sensors data and interconnect. Sensors might be obtaining a variety of input data types, including distance, light, temperature, proximity, pressure, etc. Interconnect implies that data of various types, mentioned above, may be communicated, including speech, audio and video. The front end's output provides a representation of the raw signal in a digital format, which is immediately stored on the **incident stimuli priority stack**. This stack allows to assign priorities to stimuli and process the most urgent ones at a given moment of time.

5.2.2. Pre-processing. The system is then proceeding to the next stage, **pre-processing**. During this stage, initial information estimate is performed, including *context extraction* [1], and *source identification*, which figures out where (what entity, place, etc.) the input signal is originating from, as well as the stimulus *disambiguation* with the help of the *expectation* module.

```
pre_processing
input: stimulus internal representation
output: stimulus_type, stimulus_context, stimulus_importance
```

The context extraction provides any context information that can be acquired in association with the given stimulus:

```
context_extraction
input: stimulus internal representation
output: context representation
```

The source identification is aimed at determining which source (entity, environment, etc.) the stimulus is originating from:

```
source_identification
input: stimulus representation, knowledge state
output: source id, source description
```

Other important processing elements during this stage include *sensors analysis* and *parsing*. The latter includes various image processing and natural language processing methods, while the former allows for the data from physical sensors to be collected. This data is important, as it helps

[1]The concept of contexts, as well as Active Context Stack are described in detail in Chapter 4.

the system assign priorities to various stimuli by estimating how important they might seem, e.g. “is the audio signal louder than normal?”, “are some video fragments showing signs of violence?”, “are physical sensors and external conditions OK and safe?”, “are any living entities threatened?”, etc.

```
sensors_analysis
input: stimulus representation
output: sensors data
```

It is important to differentiate functional elements presented in the pre-processing stage of workflow from the corresponding elements in the main workflow execution stage. As an example, consider parsing.

```
parsing
input: stimulus representation
output: parse trees, POS info
```

Parsing occurs in both, the pre-processing and main stage and serves the same purpose of providing certain evaluations of textual stimulus. It is done in different fashions, however, the pre-processing version lightly touches on processing the stimuli. The objective of the pre-processing version is to quickly obtain certain readily available information that would allow to assign priorities to stimuli and extract some crucial information necessary to proceed. At this point in the processing iteration, a system can push the current stimulus on a priority stack and pop a more important task (stimulus to process) from the top of the stack.

5.2.3. Current or new context? The next stage of the diagram is schematically presented with a multiplexer (data selector), an element that, given a single input and a selector function S, selects a single data output. In this case the data selector decides whether the stimulus being processed has anything to do with a context that is already known to the system, or, alternatively, it is a brand-new context that needs to be introduced. At this point there are two possible paths that the workflow can continue with, which are merged back into a single flow of execution after a few independent steps. In case of the new context, a more thorough processing of the stimulus is performed, with the data of the pre-processing step utilized to aid this analysis. Various *symbolic methods*, *semantic analysis* and *parsing* methods are utilized to achieve this goal.

```
current_context
input: context information, knowledge base
output: boolean value
```

5.2.4. Context and knowledge state updates. Once the processing is complete and it is determined whether the system is dealing with a current context, new **Active Context Stack (ACS)** frame[2] is created and initialized with the information obtained from previous stages. Relevant parent, sibling and children frames (if any) are linked with the newly created frames.

Consider now the alternative scenario when the data selector decides that the stimulus is associated with some contexts that were already introduced in the system earlier. The system then takes on determining precisely which contexts it will need to deal during present execution cycle. The corresponding ACS frames are updated and new parent and/or children connections are introduced.

At this point the system is leaving the information and stimuli processing stage, moving on to the

a) internal state updates,

b) decision making of how to react to the stimulus, and

c) performing necessary actions (system's response to the stimulus).

5.2.5. Self-model update. Once the corresponding ACS and AOS steps are over, the **Self-model Update** performs the necessary adjustments to the status of various knowledge state elements with the newly obtained information:

```
self_model_update
input: sensors data, knowledge base
output: updated knowledge base
```

These elements include the system's utilization (how heavy the workload of its hardware is), progress towards set of goals (whether progress to reach the goals is done and how fast the convergence rate is), log (document all the events that occurred during this processing cycle) and spatial dynamics, which tells the system about its status and change in spatial location in the surrounding environment.

[2]See Chapter 4 for details on ACS and AOS structure and organization.

5.2.6. Objectives derivation. Further, an unambiguous objective function is extracted and generated by the system. There might be potentially many objectives that are competing with each other and that might have different goals. A trivial case that is possible at this point is to stop processing of current stimulus due to the fact that an identical objective already exists in a system. The execution would in this case continue back to the entry point and take on a new stimulus. Alternatively, this stage can potentially generate new sub-objectives, which would aim at disambiguating the present objective of interest. This element, as well as pre-processing and stimulus processing, is of utmost importance to the system and is, in fact, one of its integral parts. This element is expected to have most fraction of the entire system's resource utilization for a lot of applications of MI, including dialogue systems, ASR, NLU, etc.

5.2.7. Axiom satisfiability. Once an unambiguous objective function is identified, an **axioms satisfiability check** is performed:

```
axiom_satisfiability
input: objective function, knowledge base
output: boolean value (OK to proceed?)
```

The importance of the axiom checker and what circumstances it could lead to, if not present or not used correctly, is outside of the scope of this workflow description, and is described in Chapter 4. At this level of analysis only abstract understanding of an objective is given to the system, therefore the axiom checker used at this stage has to be able to deal with such abstract objective function formulations.

If the axiom checker allows to continue, the system proceeds to the **Planning/Learning** step of the workflow. If, however, the check results in any axiom violations, the **Exception handling** mechanism takes control, trying to understand and update the given objective function to the state that would not disagree with any axioms and guarantee successful outcome.

5.2.8. Update objective function. Moving on, if the check results in no axiom violations, the system updates the objective function with a set of goals, requirements and any relevant Active Objective Function Stack (AOS) frames that it determined relevant for the current objective function.

```
exception_handling
```

```
input: reason for exception, objective function
output: suggested solution
```

5.2.9. Planning/Learning. After the update is complete, a Planning/Learning module devizes a plan of action.

```
planning
input: current state (knowledge base), state to be reached
output: plan representation
```

Should there be no plan found, the exception handling is activated with an attempt to better understand the goals and formulate new sub-objective functions to be processed recursively. If a plan is found, a **Plan analysis** module translates the plans internal representation into a sequence of steps to be performed.

```
plan_analysis
input: plan representation
output: suggested action items
```

The reason this module is called planning/learning is due to the fact that not all tasks require planning activity in their execution. One example of an application that does not require planning, but needs to have a learning component is dialogue systems (e.g. W blocks world: it needs to be able to learn new concepts and be able to use these concepts in the future requests).

5.2.10. Final loop execution steps. In case the plan analysis module returns a clear sequence of imperative action items, they are again checked by the axiom satisfiability module (only this time the non-abstract one), and in case of a green light, CRIS performs the requested actions and proceeds to processing of the next stimulus in the priority queue. An important optimization step that happens along the way, before the plan is executed, is choosing an optimal plan among potentially many different plans.

```
pick_optimal_plan
input: collection of plans
output: optimal plan
```

If, however, some of the action items are declarative, and need further processing for better understanding of the goal, new sub-objectives are formed recursively in order to satisfy the current objective function.

5.2.11. Full Workflow Summary. Presented workflow description is dealing with processing requests in a consecutive fashion, one after another. Consideration of the workflow under this assumption should not limit a possibility of CRIS having multiple parallel processes, each working on a particular incident stimulus acquired from the stimuli stack. Note that priority stimuli stack fits well into this model, as parallel processes will be always acquiring the most relevant task from the stack at a given time instance. Such an approach will incur certain overheads spent on the objective functions, ACS and AOS frames synchronization making sure their states are not invalidated and up-to-date. However, processing the stimuli subjected to CRIS in parallel should provide faster decision making due to increased performance of the system.

In summary, the introduction of the workflow is critical for this research project, as it allows to identify the interrelations and (potentially recursive) incident rates of the elements present in the workflow. Estimation of these incident rates, as well as measurement of the resource requirements of individual elements allows to provide an estimate of accumulated resource requirements for CRIS model and, consequently, for any MI system.

Sections 5.3, 5.4, and 5.5 presented below discuss sample applications projected onto the generic workflow. Based on the example of mapping these applications it is shown how CRIS's processing and decision making for these particular examples fit onto the generic workflow shown on Figure 5.2. Since these examples are simplified instances of the generic workflow, certain elements of it are omitted in the discussion below. This does not mean, however, that such functional elements are not applicable to any other applications of Machine Intelligence.

The choice of the applications, that are referred to as the "external drivers" in the later chapters of this thesis, allows to exercise a wide range of the workflow's functional elements. The external drivers considered in the later chapters, including the W Blocks world, the Autonomous Moving Agent (AMA), and the 3-D facial recognition, model three different areas and utilize different functional elements of the generic workflow when doing so (see Chapter 7 for detailed description of such elements).

5.3. Blocks World

The block's world is simply referred to as "W" in this work. The problem of blocks world represents a system that is manipulating objects in the world according to user's requests. Every time a new request (stimulus) is received, it is being processed in order to understand what the user wants from the system and, upon successful stimulus processing, the corresponding command is carried out on a screen. In case, however, a bad command is placed or a solution to the problem cannot be found, the system announces the negative outcome and gets back to processing next stimulus. Figure 5.3 shows the system, which is finishing executing a command "stack blocks 14, 8, and 11". In this sample run boxes and pyramids of different colors and sizes are grouped into three rows. The world's state is captured at the moment when the system is finishing executing a request to "stack blocks 14, 8, and 11" on top of each other.

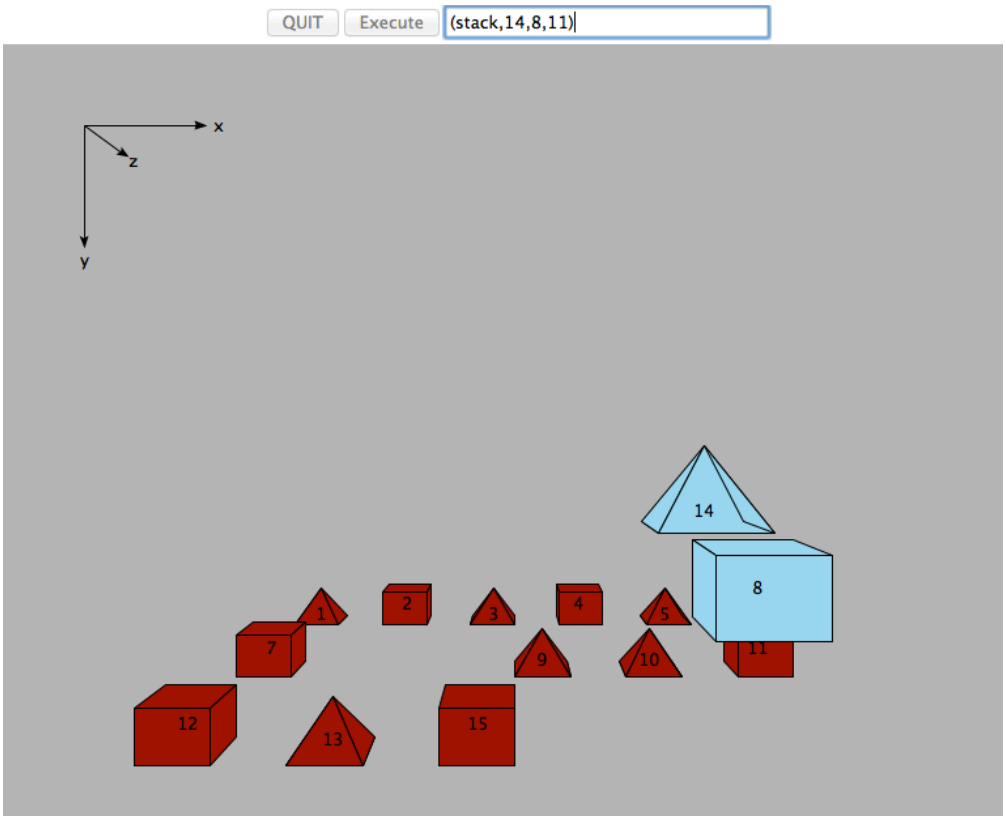

FIGURE 5.3. T. Winograd's blocks world sample run

A specific workflow that is derived from the generic workflow is shown on Figure 5.4. A sequence of steps provided below is performed when processing this request. The item numbers correspond to the numbers of the workflow elements and thus some are omitted.

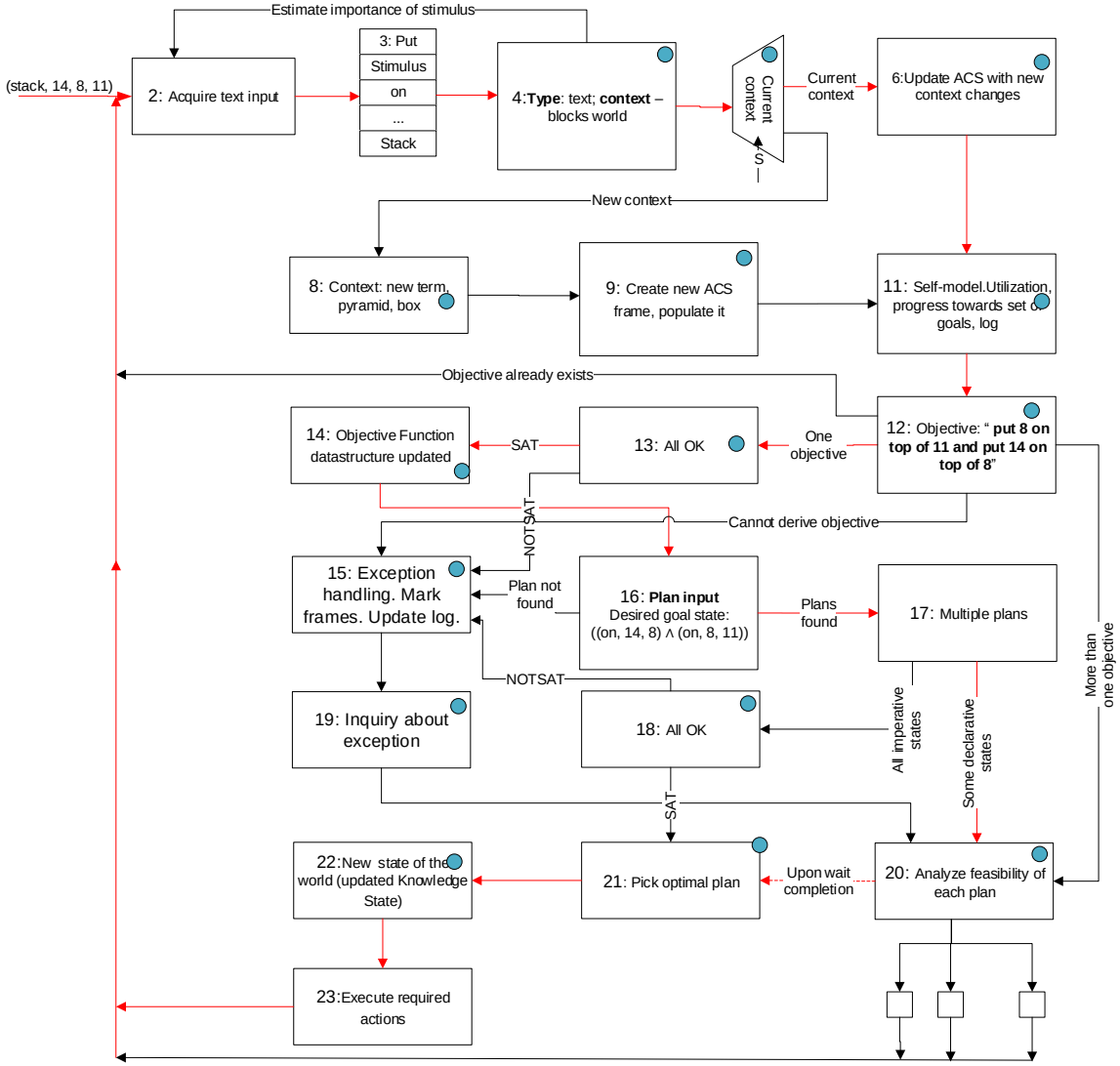

FIGURE 5.4. Workflow for the "stack blocks 14, 8 and 11" command

(1) A stimulus arrives to the system's front end.

(2) Processing begins with the front end, where the incoming stimulus is acquired in the raw form by the W machine. In case of W, the only two types of supported input obtained

from front end are text and audio signals. Both of these types should have commands or questions directed at the system.

```
front_end
input: textual stimulus "stack(14,8,11)"
output: stimulus internal representation
```

(3) The stimulus representation is put on stack.

(4) Next step, pre-processing, attempts to quickly extract any obvious information about the request type, identify any entities and their sources and estimate the urgency of such requests. In case of W the pre-processing looks as follows:

```
pre_processing
input: stimulus internal representation
output:
id objects: blocks ids: 8, 11, 14
semantics: geometrical shape, color, size of blocks
```

Context extraction, as part of pre-processing follows. In this case context extraction is determined to be as previously seen context, as the prior arrangement of blocks is supposed to be present in the world already. It is assumed that this request is performed at least after the initialization stage of the world, at which point any request would be considered to be operating in previously existing context.

```
context_extraction
input: stimulus (textual) internal representation
output: current context
```

Next, source identification module is executed. In this case the request's source is the user.

```
source_identification
input: text stimulus internal representation
output: source is the user
```

Finally, parsing is performed, as part of the pre-processing. The parsing module for W could check for syntactic errors, e.g. missing closing parathesis and other linguistic information.

```
parsing
input: textual stimulus representation
output: format of stimulus is OK
```

(5) Next, the decision making about the context is performed. For the Winograd example the decision making about the context looks as follows:

```
current_context
input: context information, knowledge base
output: new context
```

(6) Since the system is dealing with a previously existing context, ACS is updated.

(11) Next, the self model is updated. It is done in the following way:

```
self_model_update
input: stimulus data, knowledge base
output: update about system resource utilization,
progress towards goal (execute stack operation)
```

(12) Next, an objective is extracted from stimulus. For the example considered here, objective extraction is simply a parsing task that was already performed during the pre-processing stage.

```
derive_objective
input: stimulus internal representation
output: objective internal representation
```

(13) After the objective is extracted, a check is performed for any violations by the axiom checker:

```
axiom_satisfiability
input: objective function: "stack 14,8,11", knowledge base
output: OK to proceed moving straight
```

(14) An update of objective function is performed by adding the current objective to it.

(16) After the planning module is executed for W, a sequence of declarative steps to be performed is devised:

```
planning
```

```
input: current state --- blocks 14,8,11 on table,
other blocks on table;
state to be reached: (14 on top of 8) and
(8 on top of 11) and (11 on table);
output: pick 8 up, put 8 on 11, pick 14 up, put 14 on 8
```

(17) The plan analysis module in case of W example:

```
plan_analysis
input:
plan representation: "pick 8 up, put 8 on 11,
pick 14 up, put 14 on 8"
output: sequence of steps that the system understands
```

(18–23) Finally, axiom checker is performed again, plan optimization is omitted due to only one plan existing in this scenario, corresponding updates to the knowledge state are performed (new locations of objects) and the required actions are executed.

5.4. Autonomous Moving Vehicle

Consider a situation depicted in Figure 5.5, where a yellow car is the autonomous moving vehicle equipped by a machine intelligent agent operating it. In this particular situation the yellow car is finishing a passing maneuver and has an objective to return to right lane and continue driving.

A workflow diagram representation, specifically depicting this particular case of autonomous moving vehicle, is presented in Figure 5.6.

The following steps of the CRIS generic workflow are performed in order to satisfy the goal of this agent:

1) A stimulus arrives to the system's front end.
2) The front end element obtains information from the I/O devices and sensors. For the AMV example this means that all available information is collected from the system's physical front end, including, but not limited to, visual (video and picture), audio (potential commands of a human or another agent in cabin as well as external audio signals, such as honking or thunder, sensors (proximity sensor, wheel spinning sensor, velocity, trajectory,

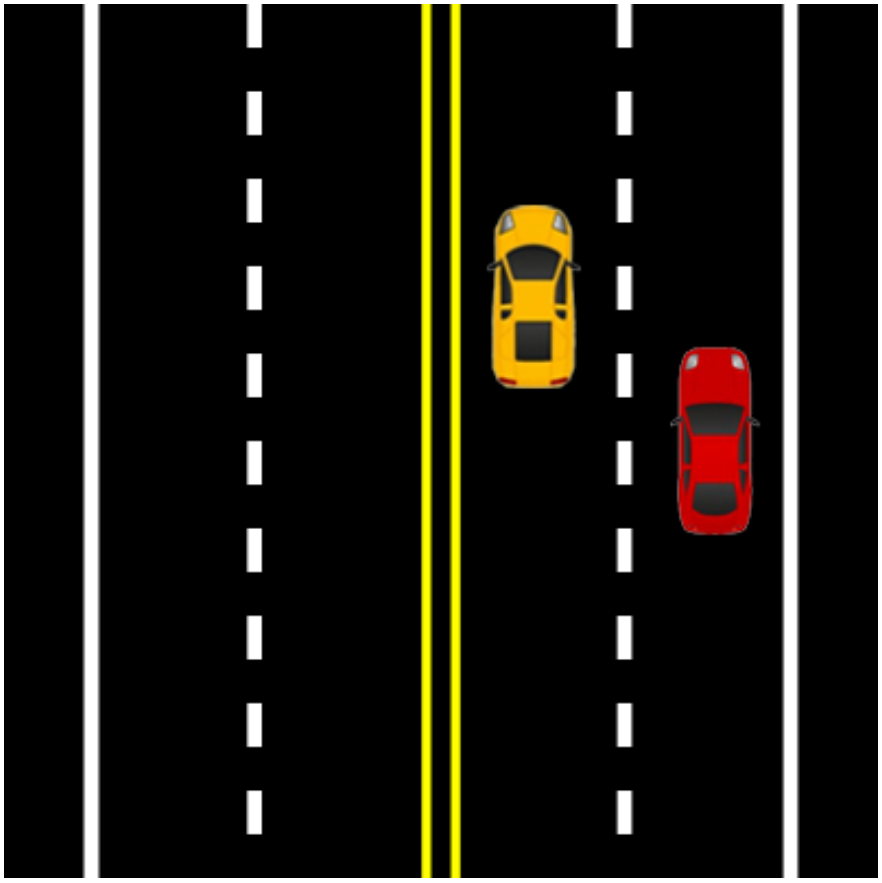

FIGURE 5.5. Autonomous moving vehicle example

temperature, etc.), as well as any remote communication stimuli over the interconnect, e.g. the AMV might be getting directions and updated tasks remotely. All the captured stimuli internal representation is the output of this function. The corresponding functional representation of front end for the AMV example is as follows:

```
front_end
input:
video, picture, audio, sensors
output: stimulus internal representation
```

After the front end module is done with current stimulus processing, it is placed on the incident stimuli priority stack.

3) All instances of captured data from front end are stored on the stimulus priority stack.

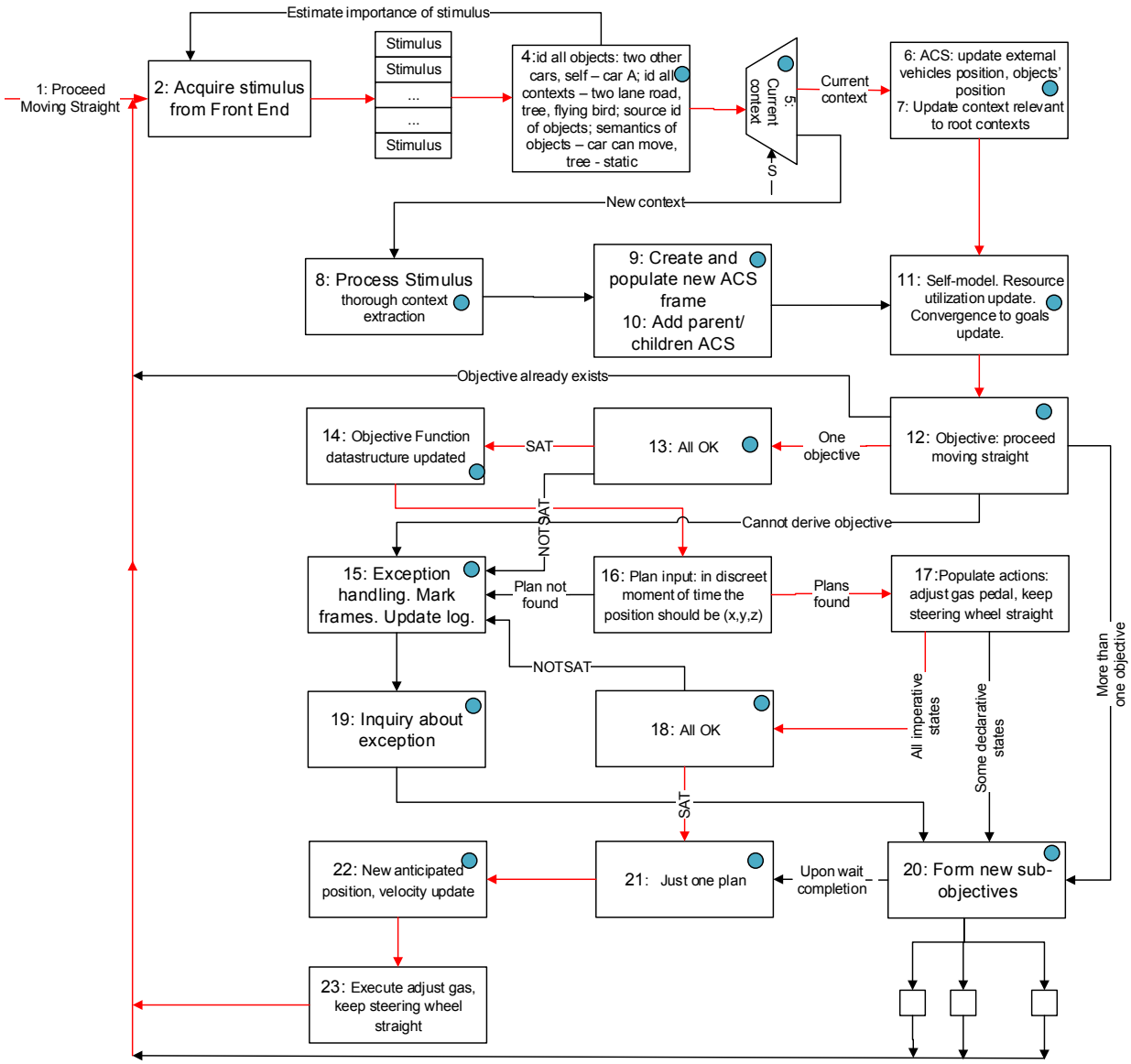

FIGURE 5.6. Autonomous moving vehicle workflow

4) The pre-processing step is concerned with collecting initial information about the stimulus, as well as the surrounding environment and any potential contexts linking objectives to this environment. Pre-processing for AMV looks as follows. Context extraction, provides any context information that is related to the given stimulus. In case of the current example, the previously existing context is obtained, which includes another vehicle of red color, the road with lane markups, scenery, including any trees and terrain.

`pre_processing`

```
input: stimulus internal representation
output:
id objects: two-lane road, red vehicle A to the right,
road markup
semantics: vehicle A can move, turn, change lanes,
road is static, road markup defines driving rules


context_extraction
input: stimulus internal representation
output: old context (red vehicle, road markup, terrain)
```

Another module executed during this stage is sensors analysis, which collects data from physical sensors.

```
sensors_analysis
input: stimulus representation
output: current velocity, proximity to the red car,
        trajectory, temperature, wheels spin
```

The parsing module that is executed as part of preprocessing is as follows. In this scenario, if an immediate importance of the audial command/question can be estimated, the current execution can be preempted in favor of a more demanding request from an external entity. It allows to establish an immediate emergency and give priority to a more demanding task to be processed immediately.

```
parsing
input: audio or textual stimulus representation
output: imperative (easy) commands/questions
```

5) A decision is made that the context was previously seen to the system, which means that it is current.

```
current_context
input: context information, knowledge base
output: boolean --- the context is current
```

(6,7) Existing ACS is updated with new information.

(11) The self-model updates of the AMV system are done as follows:

```
self_model_update
input: sensors data, knowledge base
output: update about system resource utilization,
state of hardware, progress towards goal (finish maneuver),
sensors state, physical
characteristics: velocity, position, trajectory
```

(12) The AMV objective derivation follows. In this case the objective is to proceed moving straight until safe to go back to right lane.

```
derive_objective
input: stimulus internal representation, knowledge state
output: continue driving straight
```

(13) In case of AMV the axiom checker is as follows:

```
axiom_satisfiability
input: objective function: "proceed moving straight",
knowledge base
output: OK to proceed moving straight
```

The AMV example does not imply an exception occurring under given circumstances (the vehicle is supposed to move straight without any incidents or blocking factors), so the functional representation for these elements is omitted. Potentially, however, exceptions could occur at any time, due to a flat tire, an animal on the road, etc. for the AMV case. The system therefore should always be ready to handle such scenarios.

(14) An update of objective function is performed by adding the current objective to it.

(16) Next the planning is executed. This looks as follows:

```
planning
input: current state (knowledge base),
state to be reached: travel x meters forward in 1 second
output: proceed with certain velocity forward
```

(17) The plan analysis module in case of AMV example:

```
plan_analysis
input:
plan representation: "proceed with certain velocity"
output:
keep steering wheel straight,
continue with certain velocity (calculated)
```

(18–23) Once the axioms are checked again, the knowledge state is finally updated with the current AMV's velocity, location and the required steps of maintaining the steering wheel and velocity are executed.

5.5. Generic CRIS application

Finally, consider a situation of generic CRIS application, which is again equipped with an intelligent agent that is theoretically capable of operating **any** Machine Intelligent application, including such examples as an intelligent coffee maker, an autonomous agent exploring planets (e.g. autonomous intelligent lunar vehicle) and a smart home monitoring system. Although a generic CRIS case is discussed below in application to the generic CRIS workflow, these 3 orthogonal examples will be used as special cases of generic Machine Intelligence. The following steps of the CRIS generic workflow are performed in order to satisfy the goal of generic CRIS agent:

(1) A stimulus arrives to the system's front end.

(2) The front end element obtains information from the I/O devices and sensors. In a generic CRIS case that means any audio, video, sensor and remote communication information. For all 3 example applications that implies that the information is collected from the the system's physical front end, i.e. whether any input to brew coffee has been provided (either speech or a press of a button) for the coffee machine, any remote command given to the lunar vehicle or any audio (either local or remote) command has been placed to the smart home CRIS. According to the workflow diagram on Figure 5.2, obtained stimuli are stored in a incident stimuli priority stack.

```
front_end
input:
```

```
video, picture, audio, sensors
output: stimulus internal representation
```

(4) The pre-processing step is concerned with collecting initial information about the stimulus, as well as the surrounding environment (physical and logical) and any potential contexts linking objectives to this environment. For example, in case of lunar vehicle, the system needs to identify that requests from a remote location over interconnect need to be associated with some contexts in the physical proximity of an agent. Also, during this pre-processing step, a stimulus importance level is evaluated and the stimuli is placed into the corresponding place in the stimuli priority stack. Again, this is necessary in order to attempt to process critical situations first, which potentially would lead to avoiding dangerous situations or dealing with most critical goals for the system.

```
pre_processing
input: stimulus internal representation
output:
id objects, semantics of objects,
id contexts, stimulus priority
```

The next step, context extraction, provides any context information that is related to the derived stimulus description. At this stage, the system makes a decision whether this context was previously introduced to the system workflow, or it is a brand-new context that the system is dealing with. Imagine an example of a smart home, where a user is communicating to CRIS remotely and a query of what the temperature value is inside the house is was placed. That would be a new context to the system. Now, if a user asked to decrease the temperature value by 10 degrees, that would make the system work with previously existing context, i.e. the conversation about the house temperature. The system would also have to be aware of any other metrics about the house in case the request would have to deal with them. All of those metrics would be previously existing context frames, too.

```
context_extraction
input: stimulus internal representation
output: new/old context
```

In addition to context extraction, source identification is performed as part of the pre-processing. This is necessary in order to keep track of the source id, where the stimulus was originating from. The importance of this step is obvious, as it would allow to communicate with the source in case the system is not capable of processing the stimulus or executing the objective in the form that it was given.

```
source_identification
input: stimulus internal representation
output: source is the external world
```

Another module executed during this stage is sensors analysis, which collects data from physical sensors. For lunar vehicle, this could be sensors data similar to AMV example, while for the smart home these could be any data related to the state of the house, e.g. temperature, humidity, what devices are turned on and operating, current energy consumption, CO2 levels, quality of the air inside, etc. Coffee maker sensors data could provide the coffee temperature, as well as wether buttons are pressed on the machine.

```
sensors_analysis
input: stimulus representation
output: sensors data
```

Just as the case was with specific driver applications of MI, a generic CRIS application will execute parsing as part of its pre-processing. This is done in order to be able to readily deal with trivial requests as well as to help identify those critical stimuli inputs to the system.

```
parsing
input: audio or textual stimulus representation
output: imperative (easy) commands/questions
```

(5) Next step is the decision making about the contexts associated with the current stimulus. This step is utilizing information obtained from the pre-processing step in order to make a decision whether the old context or the new context branch of the execution workflow is to be performed.

```
current_context
input: context information, knowledge base
```

```
output: boolean
```

(6, 7) The self-model updates of the generic CRIS system include any updates to the internal state in the light of the newly obtained information. These include a variety of sensors data (including velocity, physical location, external and internal conditions, etc.), internal resource utilization (how heavy the workload on the system is), "health" status of the system's hardware components, history of itself and progress made towards set of goals.

```
self_model_update
input: sensors data, knowledge base
output: updated knowledge state: resource utilization,
state of hardware, progress towards goals,
sensors state, physical characteristics
```

(12) Next is a step of the objective derivation and consequently formulation of objective function, which is further placed on the Objective Function Stack. In case of the generic CRIS application, objective derivation can become a complex step that would require substantial amount of resource due to potentially recursive nature of this derivation. As can be seen on Figure 5.2, this task might formulate new sub-objectives recursively, at which point the derivation of the initial objective will depend on potentially multiple workflow iterations. As an example, consider the smart home application. A user calls CRIS on a phone and asks to "make it feel nice inside the house before the owners and guests arrive". The system needs to derive the main objective—maintain pleasant environment inside the house (which is probably stored somewhere in the system's log, since the user is referring to this information)—which depends on a few sub-objectives: understand what pleasant environment is (query previous history of communication with that particular source of stimulus) and attempt to figure out when guests and owners are arriving (track owner's GPS location, check calendar of the event).

```
derive_objective
input: stimulus internal representation, knowledge state
output: imperative/declarative objective(s)
```

(13) A system further needs to check whether an objective of interest satisfies axioms stored in the system. In case of a lunar vehicle, for example, a violation of the axioms would be

actions that would lead to harm made to humans or other autonomous agents operating in the environment. However, axioms might be defined differently depending on a type of application in consideration, and might be less or more relaxed about the allowed actions.

```
axiom_satisfiability
input: objective function, knowledge base
output: boolean value
```

(14) An update of objective function is performed by adding the current objective to it. This means that the AOS data structure inside the Knowledge State is updated by adding a new Objective Function frame on the stack.

(16) Next the planning is executed. All three sample applications will have to involve some degree of planning in carrying out their objectives. Lunar vehicle would be similar to AMV example above, the smart home would have to plan enabling or disabling any certain equipment, and maintaining certain conditions inside the house, while the coffee maker would have to plan the proportion of coffee/sugar/water/milk in the drink. This looks as follows in a generic case:

```
planning
input: current state (knowledge base), state to be reached
output: steps necessary to reach the goal state
```

(17) The plan analysis module analyzes steps derived by the planner (if any). This step also might potentially turn to recursive dependencies, which would be very important to estimate in terms of algorithmic complexities and overall resource requirements in further chapters.

```
plan_analysis
input: plan representation:
output: sequence of steps (imperative or declarative)
```

(18–23) Once the axioms are checked again, the knowledge state is finally updated with the current agent's state, location and the required steps are executed.

5.6. Summary

This chapter introduces one of the most important elements of a model for Machine Intelligence—the generic Workflow. This workflow provides extensive amount of details on how the interaction of elements of Abstract Architecture for MI (Chapter 4) occurs during a typical iteration of CRIS. The discussion presents two layers of the workflow, an abstract basic workflow representation, along with the detailed workflow specification. The former allows to envision an overall picture of how the workflow is supposed to be operated, while the latter provides a clear and well-defined approach on how to implement the CRIS workflow. The level of detail and abstraction of the presented workflow model is hypothesized to be sufficient to serve as a template for modeling generic Machine Intelligence. Hence, the measurement results obtained from experiments defined by such model are true, if this hypothesis is true.

Further, a series of driver applications is introduced and a detailed discussion on how these applications map onto the generic workflow is provided. These driver applications include T. Wingorad's Blocks World, referred to as "W", Autonomous Moving Vehicle (AMV) subjected to an external driving environment as well as a generic CRIS driver application. The latter is not to be confused with a generic CRIS workflow, as it provides a description of the execution order for CRIS's Abstract Architecture functional elements. A generic CRIS driver application can in fact be considered a system of Machine Intelligence by definition, as it utilizes the generic Abstract Architecture as well as generic Workflow in order to define itself. Generic CRIS driver application is an abstract machine that is capable of executing **any** Machine Intelligent task. Without loss of generality, a number of examples of advanced MI applications are provided in order to show that this system is generic.

Now that the two main aspects of a system for Machine Intelligence, namely Abstract Architecture and Workflow, are introduced, next chapter will be dedicated to the discussion of a Performance Model and Metrics for MI.

6

Performance Model for Machine Intelligence

This chapter introduces the final required piece of the model that enables estimating the lower bound of resource requirements for problems of Machine Intelligence (MI): the Performance Model. Previous chapters introduced two other important aspects of MI: the Abstract Architecture and the Workflow, shown in Figures 6.1, and 6.2. The abstract architecture provides a detailed definition of all components of the Cognitive Real-time Interactive System (CRIS), which is one possible system of MI and the focus of this thesis. The workflow provides details of a step-by-step execution for any problem of MI, with various potential scenarios and outcomes (for detailed description of corresponding concepts see Chapters 4 and 5 respectively). Now that these two major elements are introduced, the crucial missing element that is required in order to provide the full model for estimation of resource requirements for MI is the Performance Model. These three elements combined together will provide a foundation for exploring the space of all possible problems of MI. For instance, how such programs operate, and how various functional elements of the architecture interact with each other via the worklfow. Finally, in order to accomplish the goal of this work, determining what the resource requirements for MI are, the performance model introduced in this chapter will be used. The performance model, combined with metrics of driver applications subjected onto the generic workflow, will allow us to analyze, extrapolate and predict the future bounds of resource requirements for problems of MI, provided that there is some prior knowledge, such as algorithm asymptotic complexity analysis or sample runs that present a relationship between the size of a problem and metrics of interest for the specific MI problem.

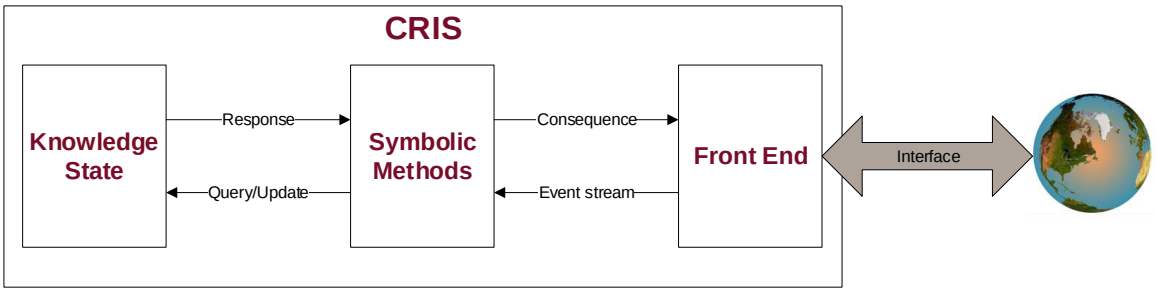

FIGURE 6.1. Recap of abstract architecture for Machine Intelligence. The abstract architecture at the top level consists of three elements, Knowledge State, Symbolic Methods, and Front End. This models a closed-loop system, where Front End is interfacing with the external world and Symbolic Methods, while Symbolic Methods are functionally dependent on and obtain data from the Knowledge State.

6.1. Main metrics for MI

The Performance Model is defined in the form of an equation that includes requirement estimates for the most significant elements of the Workflow. The formula provides estimates expressed in the amount of execution time that is estimated for an application of MI. A few examples of classes of MI applications are summarized below along with examples of how the performance model works. In addition to defining a general performance model that will serve as the foundation for defining bounds on the computational resources, this chapter provides a discussion of the metrics and parameters that can be used in order to characterize such requirements.

A metric, **Floating Point Operations per Second (FLOPS)**, and a requirement parameter, **main memory capacity**, are used in this study in order to quantify the bounds on resource requirements. Main memory capacity is a requirement because it enables a particular problem to run given enough memory is allocated for that problem. An increase in available memory beyond the required point would not affect the performance, or the time to solution of a problem. Since this study is concerned with problems of MI that operate in real time, another requirement imposed on the execution of problems considered is **time to solution**. For the applications of MI real time implies that the time to solution requirement should be on the order of seconds, normally ranging from one to three seconds. Some additional metrics that might be considered in the future to further improve understanding of resource requirements are provided below in Section 6.3.

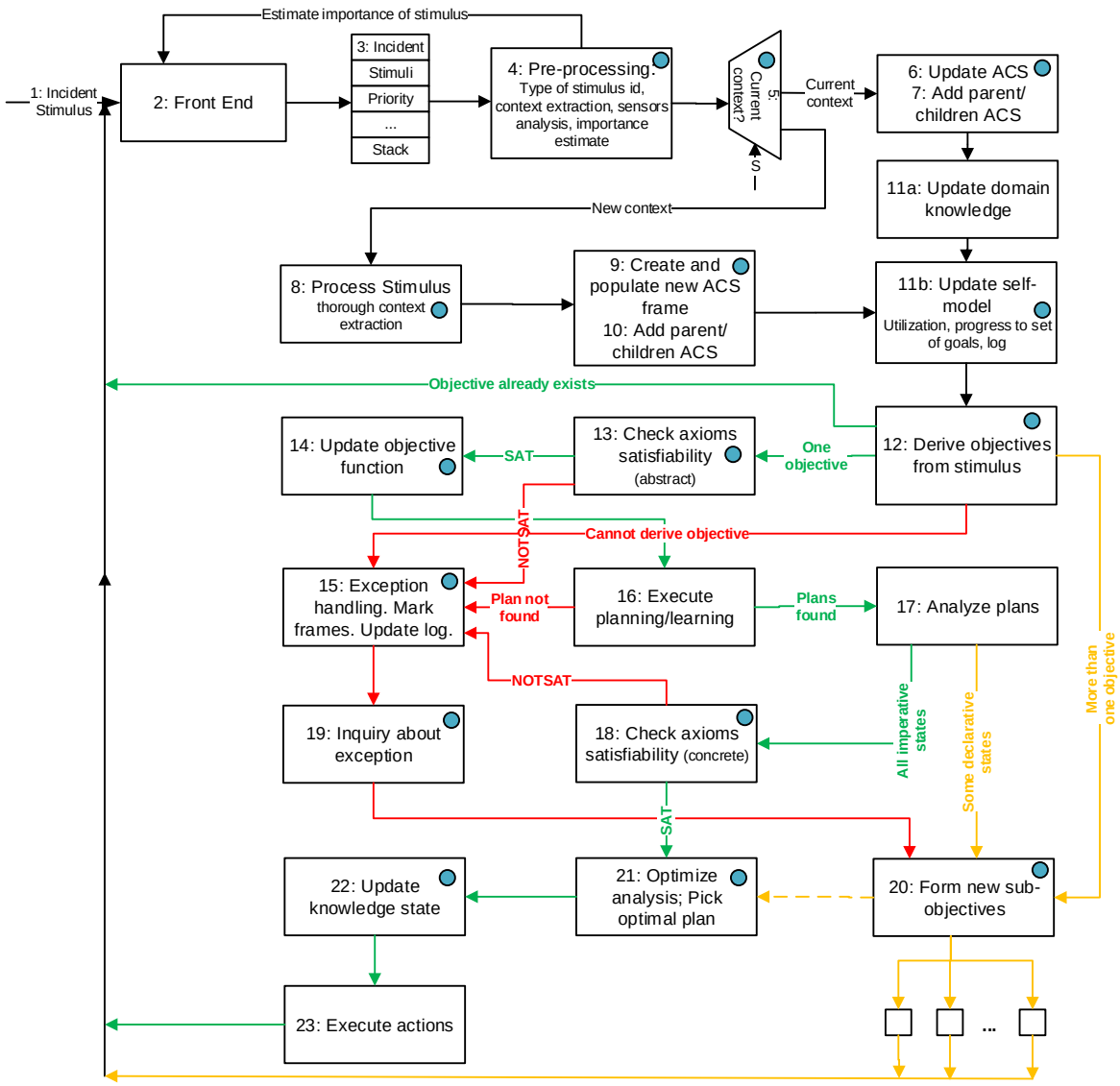

FIGURE 6.2. Recap of generic workflow for Machine Intelligence. This workflow provides a description of step-by-step execution of an iteration of any machine intelligent system. Blue dots indicate potential read/write accesses to the Knowledge State. Green traces indicate normal execution, while red traces suggest a problem, which might need to be reevaluated or might not be possible to solve. Yellow traces indicate data dependencies on solutions of future iterations of the system.

Looking at only the three metrics and parameters mentioned above as characteristics of resource requirements for problems of MI is one possible view on the problem. Such an approach is not generic, as characterizing machine intelligent behavior will, most likely, require several additional metrics that would accurately reflect all various aspects of computations involved by

providing a bigger picture. However, it is hypothesized that this approach of characterizing resource requirements of machines for certain classes of applications of MI via the proposed metrics and requirements is important and useful. Consider semantic parsing as an example application to which the metrics of interest could be applied. It is estimated that for a problem with a vocabulary size of one thousand, the number of operations required is on the order of one billion. Therefore, to run this kind of a problem in real time, one would need to have computational resources of an order of one billion FLOPS, which is a useful general metric for the MI performance model since the majority of systems and algorithms already present performance reports in it.

Further, the hypothesis that the approach chosen is important and useful finds support in the majority of scientific publications on the topic of resource utilization for applications of Machine Learning (ML) and Artificial Intelligence (AI) [**12**, **15**, **30**, **32**, **36**, **37**, **56**, **96**, **102**, **124**, **152**, **158**].

As it has been mentioned in earlier chapters, the question of resource utilization for approaches and algorithms of MI has not been extensively studied, but those documented efforts mainly are concerned with the number of **Operations (Ops)**[1], usually expressed in the form of big O notation using asymptotic analysis (see Table 6.1). Main memory was added as an additional metric for estimation of resource requirements (because it is one of the standard metrics for reporting in the Top500 and Graph500 lists). Additionally, the performance model, abstract architecture, and workflow are designed in a generic way that allows for straightforward extension to incorporate additional metrics of interest in the future. In fact, the abstract architecture and workflow would not be affected at all in the event additional metrics are introduced to the study. None of the existing metrics are driving the decisions of the architecture design or CRIS's algorithmic behavior. The performance model would be the only component that would require changes. A discussion of the kinds of changes necessary if new metrics are introduced is provided after the formula definition of the performance model (see Section 6.2 below).

As an example of a potentially useful metric for MI, consider the case of graph applications, as a subset of applications of MI. Such a subset of problems is quite large, including graph algorithms of finding shortest paths [**69**], planning [**112**], and semantic and syntactic search in Natural

[1]It is imperative to distinguish between two different metrics: Ops, and FLOPS, which have different units. The former is just an integer number that characterizes how many operations certain software or algorithm requires to execute, while the latter is the rate of that number of operations per unit of time. For simplicity and to make the two terms distinguishable, the number of floating point operations will be referred to as Ops, while the number of floating point operations per second will be referred to as FLOPS throughout this thesis.

Language Processing (NLP) [**78**, **94**, **116**]. One possible metric to express the performance of graph algorithms executed on High Performance Computing (HPC) systems is Traversed Edges per Second (TEPS). One could consider a question of finding a best class of machines (from architecture and software points of view) to run these kinds of problems. If an approach of only looking at FLOPS and memory capacity were chosen, one of the best machines to run problems of MI, and therefore graph problems, would be Tianhe-2A, which is as of this writing considered the fourth fastest cluster according to the Top500 list [**185**]. Even though this machine is great for delivering one of the best High-Performance Linpack numbers ($R_{max} = 61,444.5$ TFLOPS[2] vs 122,300.0 TFLOPS fastest machine, Summit, on Top500 list), it would not be as competitive in delivering the best TEPS numbers (2,061.48 GTEPS[3] vs 38,621.4 GTEPS which is what the fastest machine on Graph500 list, the K computer, achieves [**73**], see Figure 6.3).

The same goes the other way around, with the fastest supercomputer on Graph500 list, the K computer, being the 16^{th} on the Top500 list (10,510.00 TFLOPS). So, in this example, if a user were just concerned with FLOPS and main memory capacity to assess requirements of MI problems for graph applications, the estimates would yield a range of results varying by an order of magnitude in the worst case scenario. Figure 6.3 shows two functions, number of TTEPS, and TFLOPS achieved on the 10 most prominent machines.

The domain is sorted in the ascending order of the performance results for TEPS, with K computer being first on the Graph 500 list (as of June 2018), and Tianhe-2 finding itself at 10th place. It is observed that the FLOPS graph, however, is not monotonically decreasing, with certain supercomputers behaving unexpectedly. The best example of such inconsistent behavior in terms of TEPS vs FLOPS is Tianhe-2 system, which is the lowest out of ten in terms of TEPS, while being the second best performing in terms of FLOPS. Consequently, using FLOPS instead of TEPS in CRIS would introduce a variance in performance estimation.

In summary, three metrics and requirement parameters, FLOPS, main memory capacity, and time to solution were chosen in order to quantify the problems of MI. This was done because these are the most widely used metrics in quantifying algorithms and ranking the current fastest machines. This approach is not intended to be the final or best way to quantify classes of all problems of MI. Rather, it is intended to provide an initial approach to performance estimation

[2] T—Tera=10^{12}
[3] G—Giga=10^{9}

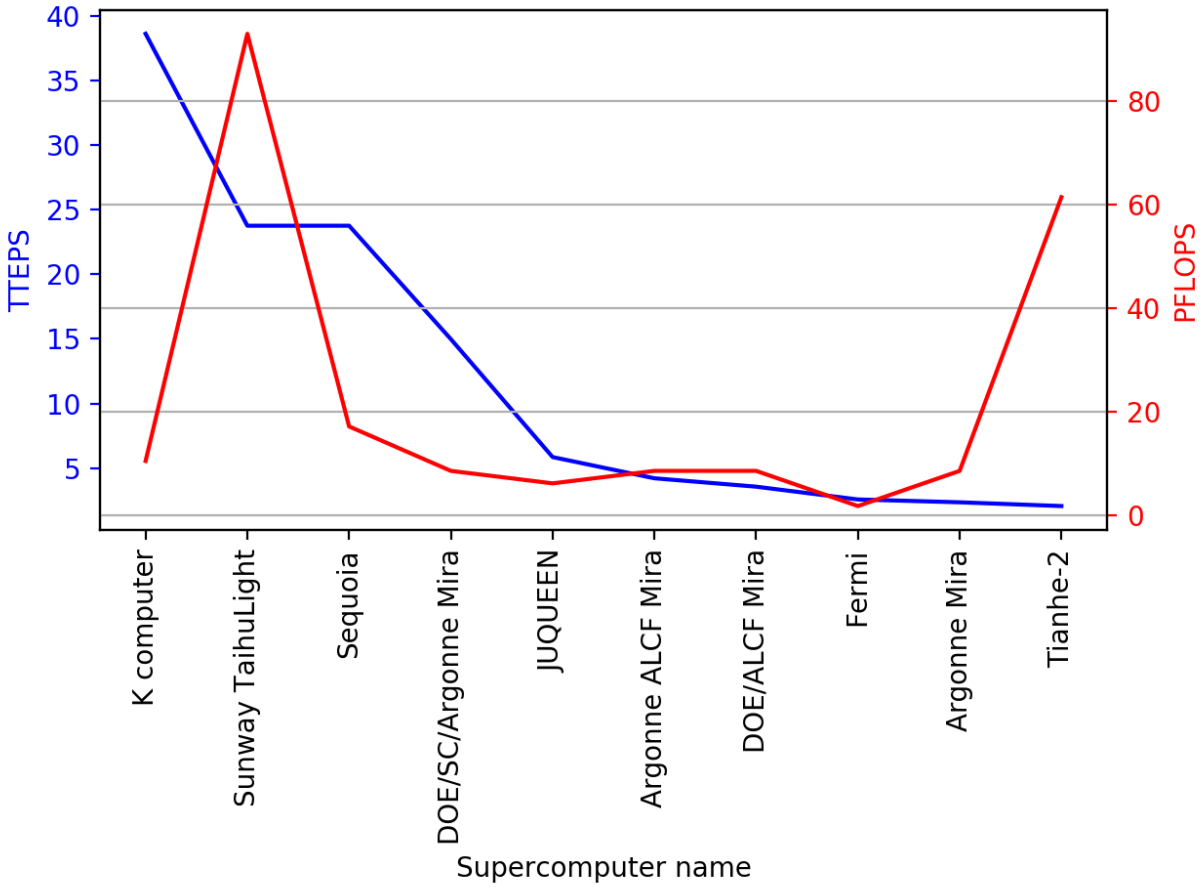

FIGURE 6.3. The amount of TEPS and FLOPS that the top ten most powerful supercomputers (according to Graph500, i.e. TEPS-wise) achieve. The domain in sorted in the descending order of achieved TEPS (from faster on the left to slower on the right). A corresponding FLOPS graph shows non-monotonic behavior. The fact that there is not a correlation between the fastest TEPS and FLOPS machines is obvious: compare the FLOPS of Sunway TaihuLight and Tianhe-2 (approx. 93 TFLOPS vs approx. 61 TFLOPS, which are both the same order of magnitude) vs the TEPS of the corresponding machines (approx. 23 TTEPS vs approx. 2 TTEPS, which observe an order of magnitude difference).

for problems of MI. An example of where this specific approach might provide wrong results has been provided. It is hypothesized that, irrespective of limitations described above, this approach is useful and is a good first approximation to use these two metrics to quantify the performance of MI. Based on the results of this research study, future approaches can be easily adjusted to include additional metrics that would more precisely express requirements for MI. Additional metrics that may be helpful in expressing resource requirements are discussed in Section 6.3 below. A brief discussion as to why these metrics may be important and beneficial is provided. The next

section introduces an important concept of the performance model, based on the proposed metrics and requirement parameters.

6.2. Performance Model

An approach to defining the formula describing the Performance Model is taken from the point of view of low-level metrics that determine requirements for MI. The three most significant metrics and requirement parameters that this research is based on are the number of FLOPS, memory capacity and time to solution that are necessary in order for systems to support problems of MI in real time.

Prior to introducing an equation that describes the Performance Model, an overview of various applications of MI along with their asymptotic computational costs is provided. Table 6.1 shows a number of such activities, their Ops estimates, as well as memory capacity that is needed in order to support these activities. Some of these estimates are taken from prior research in the field, while others are estimates and derivatives based on the known algorithms in the corresponding areas of MI. A brief description of various areas of Machine Intelligence from Table 6.1 follows. The summaries of different applications from the table are supplemented by the descriptions of problem size parameters.

Facial recognition is a technology that attempts to identify and recognize a person from a digital image or a a frame from a video source. Normally, a facial recognition algorithm will operate by comparing facial features of a face in question against an existing database of faces. Today's mainstream approaches are based on deep learning and machine learning. However, alternative approaches not utilizing ML also exist (see Chapter 7). The size n is defined as the number of points in the face point cloud, while n^2 for the traditional ML approaches is the number of pixels in an image.

Automatic Speech Recognition (ASR) is a subfield of computational linguistics, computer science, and electrical engineering that is aimed at translating spoken language into text by machines. This field represents one very important class of problems of Machine Intelligence. Various solutions, provided by major leading software companies, including Google, Apple, Baidu, Microsoft and others, find their applications in various fields ranging from personal assistants to education and help to people with disabilities. The parameter m defines a size of a vocabulary that the

Problem of Machine Intelligence	**Complexity F_i, Ops**	**Main Memory, B**	**Explanation of parameters l, m, n**
3-D Facial Recognition	$\mathcal{O}(n^3)$ **[32]**	$\mathcal{O}(n^2)$ **[32]**	n — number of points in pointcloud
Facial Recognition (ML approach)	$\mathcal{O}(n^2 \log n)$ **[30, 124]**	$\mathcal{O}(n^2)$ **[30]**	n^2 — number of pixels in image
Automatic Speech Recognition (ASR)	$\mathcal{O}(mn)$ **[12]**	140 GB **[27]**	m — size of vocabulary, n — order of singular value decomposition ($n \ll m$), therefore worst case might be $\mathcal{O}(m^2)$
Dependency Parsing	$\left[\mathcal{O}(m + n \log n) \dots \mathcal{O}(n^3)\right]$ **[96]**	1.8 GB **[33]**	m — no. of edge labels, n — no. of nodes (sentence words) of a directed graph
Semantic Parsing	$\left[\mathcal{O}(n^3) \dots \mathcal{O}(n^5)\right]$ **[96]**	3 GB **[21]**	m — no. of edges, n — no. of nodes
Autonomous Moving Agent	$\mathcal{O}(n^2)$ **[152]**	2 GB **[129]**	n — number of points defining obstacles
Planning	$\mathcal{O}(n^2)$ **[153]**	16GB **[100]**	n — number of nodes in the search graph
Lip Reading (augmented by ASR)	$\mathcal{O}(n^3)$ **[138]** (estimate)	$\mathcal{O}(n^2)$ **[138]**	n — product of number of frames/s of video for lip reading and frames/s of audio for speech recognition
Handwriting Recognition	63 TFLOPS **[66]**	4096 + 896 MB **[178]**	

TABLE 6.1. Performance model components and their asymptotic analysis. Values of l, m, and n define sizes of the respective problems. In cases where asymptotic estimates of memory or Ops were not available, known values of particular experiments are presented.

Automatic Speech Recognition (ASR) system is operating on, and n is the order of singular value decomposition ($n \ll m$). The worst case scenario is therefore $\mathcal{O}(m^2)$.

Two different approaches to parsing, which are both examples of problems in Computational Linguistics and Natural Language Processing, are in part used by systems of Speech Recognition mentioned above. Parsing, however, can be used for a wide variety of other problems, including analysis of any text, such as library collections, or social media. The parameters of problem sizes for parsers are as follows: m is the number of edges and n is the number of nodes of a directed graph, with nodes being the sentence words and edges—connections between those words.

Autonomous Moving Agent is an agent that performs tasks with a high degree of autonomy. The essence of *agency* is that "an agent can control to some extent its own destiny" [**175**]. That requires automaticity—the agent to have capability to sense the environment and to act upon it and not require intervention of any external entities [**61**]. The usual applications of such agents include space missions, terrain discovery, household maintenance, etc. The problem size n in the table above signifies a number of obstacles that the agent is subjected to throughout its execution.

Planning is a branch of AI that is concerned with an ability of intelligent systems to come up with a sequence of actions, given an initial state (most often the state an intelligent agent is in) and a goal state.

Lip reading is an example of an area where AI has shown impressive results, with automated systems beating the experienced human lip reading capabilities (93.4% accuracy vs 52.3% average across all subjects) [**9**]. The problem size n in the table above incorporates a product of number of frames per second of video stream and frames per second of audio stream.

Handwriting recognition is an ability of a computer to obtain and interpret intelligible handwritten text. Most of the time handwriting implies Optical Character Recognition (OCR).

Now that descriptions of various examples of problems in the area of MI are provided along with their Ops and memory capacity estimates, below is the formula that defines the performance model for MI:

$$T = \sum_{i=1}^{N} C_i T_i + \sum_{i,j=1}^{N} C_i C_j M_{ij} T_i T_j, \tag{6.1}$$

where T is the overall time in seconds that is required for a specific problem or a class of problems to execute, $N > 0$ is the number of problems considered, T_i is the expected time to solution in

seconds for a particular problem, $M_{ij}, i, j = 1, \ldots, N$ is a matrix that signifies whether the objective functions of problems i and j are associated and have to be worked out together, while $C_i, i = 1, \ldots, N$ defines combinatorial features associated with particular MI problems. In order to use Equation 6.1 with Ops estimates from Table 6.1, it is important to note that the expected time to solution for a problem i is $T_i = F_i/R$, where F_i is the number of Ops from Table 6.1 for a specific problem i, and R is the achieved rate measured in Ops per second (or FLOPS) on a certain machine for problem i. Using the definition of T_i, equation 6.1 can also be expressed in the form:

$$T = \frac{1}{R}\sum_{i=1}^{N} C_i F_i + \frac{1}{R^2}\sum_{i,j=1}^{N} C_i C_j M_{ij} F_i F_j. \tag{6.2}$$

To understand how these equations work, consider an example of the autonomous moving agent augmented with a task of facial identification and 3-D recognition whenever a face is believed to have been identified in an immediate vicinity of the agent.The intent of the performance model is to estimate the resource requirements (either in FLOPS or in seconds) of this activity. According to the Table 6.1 the autonomous moving agent and the 3-D facial recognition should incur the values of coefficients provided in Table 6.2.

Coefficient	**Autonomous moving agent ($i = 1$)**	**3-D facial recognition ($i = 2$)**
n_i	$n_1 = 20$	$n_2 = 4500$
C_i	$C_1 = \mathcal{O}(10^1)$	$C_2 = \mathcal{O}(10^3)$
M_{ij}	$M_{12} = 1$	$M_{21} = 1$
F_i	$F_1 = \mathcal{O}(10^2)$	$F_2 = \mathcal{O}(10^{10})$

TABLE 6.2. Values for parameters of two problems of MI: the autonomous moving agent and the 3-D facial recognition.

The values of n_1 and n_2 are chosen as follows: $n_1 = 20$ signifies an example with 20 obstacles in the external environment, while $n_2 = 4500$ is the average number of points of 3-D point cloud obtained from 20 cameras. The coefficient $C_1 = 10$ signifies that not only a single operation of planning occurs throughout the system's execution, but 10 alternative plans are considered as an example. The value of is calculated as follows $F_1 = \mathcal{O}(n_1^2) = \mathcal{O}(10^2)$. In a similar fashion, suppose the 3-D facial recognition algorithm uses a database with one thousand faces which yields: $C_2 =$

$\mathcal{O}(10^3)$, $F_2 = \mathcal{O}(n_2^3) = \mathcal{O}(10^{10})$. Finally, since the two methods, the autonomous moving agent and 3-D facial recognition are supposed to work jointly, $M_{12} = M_{21} = 1$. Plugging all of the above terms into the Equation 6.2, we get $T = \frac{1}{R}[C_1F_1 + C_2F_2] + \frac{2}{R^2}C_1C_2M_{12}F_1F_2$. Now, if it is assumed that these computations were to be performed on one compute core of a conventional MPP (Cray XE6, the BigRed II MPP), which delivers approximately $R = \mathcal{O}(10^{10})$ FLOPS for certain applications of MI, the results of that combined task would be achieved in

$$T = \frac{1}{\mathcal{O}(10^{10})}\left[\mathcal{O}(10^3) + \mathcal{O}(10^{13})\right] + \frac{2}{\mathcal{O}(10^{20})}\mathcal{O}(10^{16}) = \mathcal{O}(10^3)$$

seconds. If the same problem were to be executed on a configuration with $R = \mathcal{O}(10^{12})$, the execution time $T = \mathcal{O}(10^1)$, which is real time processing on the order of one second. If all available resources of BigRed II ($R = \mathcal{O}(10^{14})$) would be used, it would require $T = \mathcal{O}(10^{-1})$ seconds to run the problem considered.

Following an approach similar to the one presented above, the formula for performance model can be used in order to determine resource requirements for any combination of tasks of MI. Since this research is situated around tasks of real-time processing, given the number of Ops for a particular combination of tasks, the performance characteristics of the machine capable of processing such tasks in real time can be obtained.

6.3. Additional metrics

This section discusses alternative potential metrics that can be used to express resource bounds for problems of MI. In addition to the three metrics and requirement parameters introduced and discussed above in Section 6.1, the additional metrics described below can be useful when quantifying the resource requirements. These metrics have not been included into the present performance model, as the present approach defines the performance model to the first order approximation by including the most significant metrics in it. The additional metrics to be considered are:

1) Measures of intelligence. How "smart" or "intelligent" the system is at conducting various tasks of MI. Separate consideration is needed to define the units of such a metric, as well as how to measure the values of the metric. An approach chosen to measure intelligence should be robust, without shortcomings inherent to some existing approaches in the area

of AI. Consider an article by Su et al. [**180**], where the authors are exposing shortcomings of particular deep learning algorithms by altering a single pixel in test data. The results for some of the test data classification that are obtained under such scenarios are sometimes astonishing, with the systems predicting an object in the test data to be semantically a completely different entity (e.g. an airplane instead of a deer) with confidence level well beyond 85% (for some cases 99%).

2) Size of information storage. Simply put, this measure is concerned with the total number of bytes that the system of MI is operating with and has the capacity of. This is different from main memory capacity in that this measure will quantify the amount of data read and written from disk storage. It, therefore, falls a few orders of magnitude behind the main memory characteristics of latency and bandwidth.
3) Throughput. How fast would the information have to be transmitted from storage locations (potentially in the cloud) to a particular CRIS system. This metric is measured in bytes per second and could be applied to either a local MI system (i.e. the disk storage or main memory) or processing of data that involves network accesses. CRIS entities can operate through a network maintaining and constantly updating a state of universal knowledge. These updates can be reflected in a centralized registry and later be used by other entities upon request.
4) Operations primitives. This is a higher level set of metrics that is not inherent to the hardware system that the problem is run on. The primitive operations include input (audio, video, sensors data), output, updates and queries of the knowledge state. These higher level metrics can be used in order to quantify the applications of MI and hardware systems supporting such applications in a way similar to the metric TEPS quantifying modern supercomputers for the problems of graph processing. Measures of the rate of parallel input per second (e.g. multiple input video and audio channels being processed concurrently) or updates of a system state per second may be beneficial for quantifying problems of MI.
5) Concurrency. The concurrency can help to estimate the number of simultaneous operations a system would be able to execute. This metric will have to take into account a considerable amount of previously mentioned parameters and, essentially, represents a combination of them.

6) Data movement channels.

   When considering quantification of a system for MI, a few various types of communications are distinguished. All of them will play an important role in measuring the overheads and latencies the system would encounter in performing specific computations. The types of communication can be classified as:

   - System-wide communication
   - Communication to relatively local memory
   - Intra-chip communication
   - Broad external I/O and mass storage

7) Energy and power. This is another really important metric that can be incorporated into the performance model in the future. It is important because energy consumption is one of the crucial aspects when it comes to almost any application of autonomous or mobile agents possessing elements of MI. With significant attention already paid to delivering not only the fastest, but power efficient (FLOPS per watt) systems [**74**], this metric will most probably be one to incorporate into the performance model.

8) I/O channels will present additional metrics, such as:

   - Knowledge access rate. This metric can provide an insight on the rate of I/O when generating or accessing the knowledge state of CRIS.
   - How many channels are there in each I/O category
   - Characteristics of each I/O channel: speech, quality of speech, throughput
   - Interconnect
   - Sensors. Temperature sensors are an example.
   - Physical + tactile manipulation

6.4. Summary

This chapter introduces one of the key elements that combine efforts of resource requirements estimates for problems of Machine Intelligence—the Performance Model. The model is presented in a form of a formula that takes into account metrics and requirements, which are the number of FLOPS, time to completion, and main memory capacity. A substantial number of additional potential metrics are presented and discussed. It is argued that such metrics could easily be added to

the performance model in the future without affecting the actual model that defines a system for MI. This thesis is now proceeding to the last part of its logical partition, the description of experiments, and consequently reporting of the results. Next chapter, Chapter 7 provides description of the external driver applications that are mapped onto the workflow. Chapter 8 discusses results that were obtained by executing these experiments. Finally, Chapter 9 provides estimates on the lower bound resource requirements for MI, and Chapter 10 presents the conclusions, future work and final thoughts.

7

Description of Experiments and Methodology

This chapter presents an overview of an experimental setup that had been put in place in order to obtain the results provided and discussed in the next two chapters. Driver applications, mapped onto the functional elements of CRIS architecture, following the principles of generic workflow, are presented and discussed in details below. Three driver applications are considered: i) Wingograd's blocks world, ii) 3-D facial recognition, and iii) Autonomous Moving Agent (AMA). Brief summaries and motivations are provided for each external driver prior to the actual experimental setup descriptions. In addition, it is shown how each of the drivers is related to the generic workflow of Machine Intelligence (MI) (Figure 5.2).

7.1. Blocks World

This section introduces an external driver, the blocks world or SHRLDU initially presented and implemented by T. Winograd [**196–198**]. This driver was chosen as a simplistic analog of a generic architecture and workflow of Cognitive Real-time Interactive System (CRIS) (see Figure 7.1) and as an experimental base in order to practice and test some fundamental ideas of the system. Such an exercise presents an important set of problems that are imperative and relevant to both the blocks world system as well as the generic workflow, including task planning, natural language understanding and processing along with knowledge management and representation. Some of the approaches implemented within the framework of this exercise provided better understanding of the interrelationship of functional elements in CRIS. Before proceeding with the details of the driver application, a brief survey of Winograd's original SHRDLU system is provided.

In the end of 1970s T. Winograd had developed a program for understanding natural language, called SHRDLU. This program carried out a simple dialog with the user about a small world of

objects (called BLOCKS world). The main focus of this work was on the language parser that allowed the user to interact with the machine using English natural language as input. The user instructed the system to move various objects in the blocks world. Among the most important features, which at the same time allowed for such a system to appear intelligent to a human, were:

1) the blocks world was simple and limited; the entire set of objects and locations could be described by a lexicon containing about 50 words;
2) the system retained record of actions performed in the context of the conversation;
3) due to keeping track of the history of requests and tasks performed, the system could answer questions of the sort "Can you put a box on a pyramid?" or "Can you stack two pyramids together?".

The choice of Winograd's BLOCKS as a driver application for the workflow allowed for a better understanding of common requirements and practices needed for implementation of interactive systems analogous to CRIS, but with the reduced semantics and functionality. Most importantly, this implementation was used to obtain measurements of bounds on resource requirements. As will be shown in Chapter 8, the real time regime (sub one second) is achieved for certain number of blocks in the world.

Queries of the sort "put block X on top of block Y", "stack blocks X, Y, and Z" were implemented. Additionally, question answering scenarios were implemented, including types of questions "did you pick block X up?", and in case of the positive response, asking "why?", which would generate an answer explaining why a certain block was picked up (e.g. as an objective requirement in order to complete a certain rearrangement of the blocks). Additionally, experiments with integrating an Automatic Speech Recognition (ASR) into the blocks world workflow were carried out [**65**]. In order to obtain the resource requirements of the blocks world, the input data from the AIPS-00 planning competition for the blocks world was used [**2**].

Figure 7.1 below shows details of how the blocks world is mapped onto the generic workflow. The red arrows in the Figure indicate a route that is taken in order to process the text stimulus of "(stack, 14, 8, 11)", which means to stack three objects with numbers 14, 8, and 11 on top of each other respectively. A brief description of how main processing modules of the workflow operate follows.

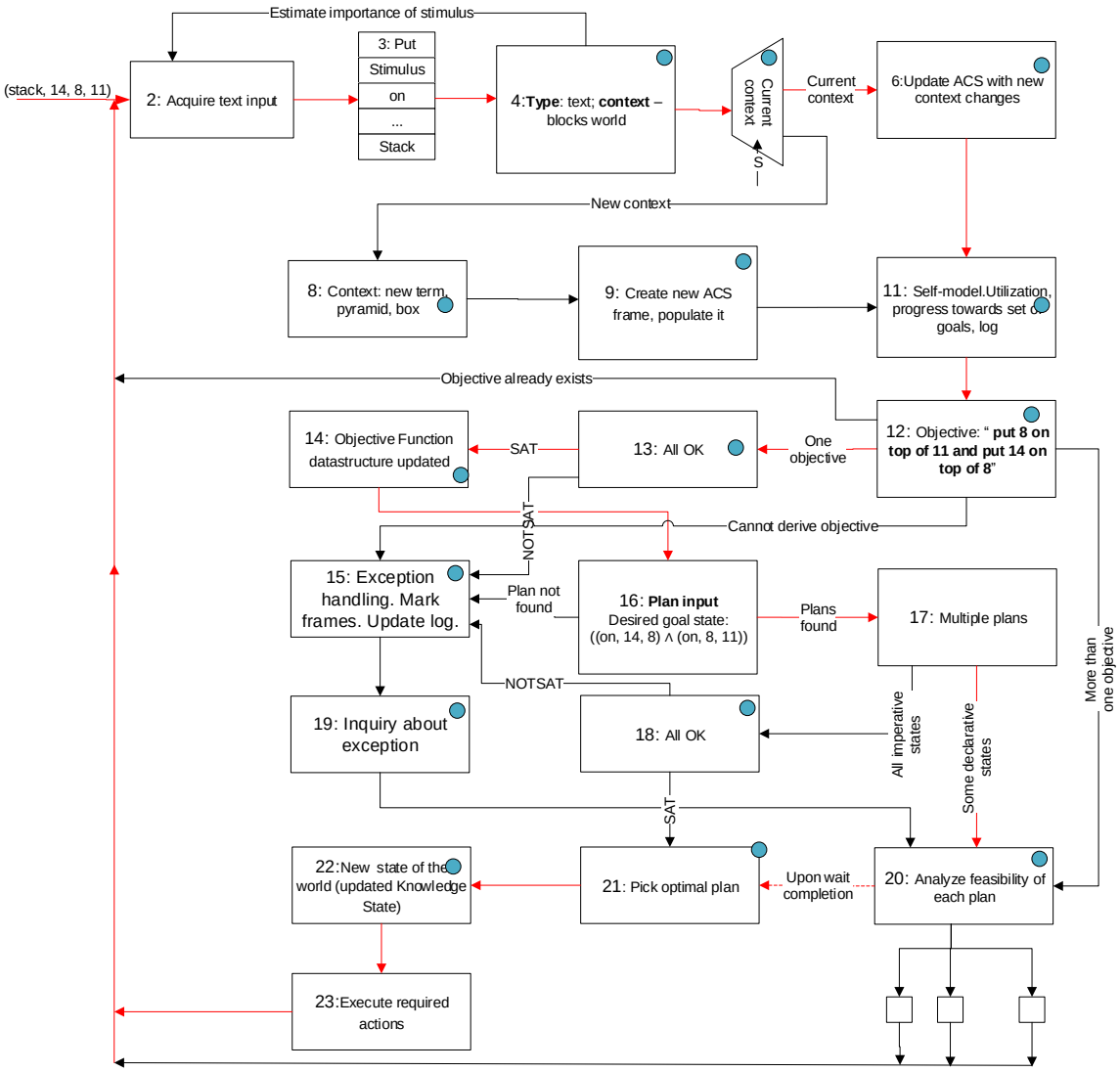

FIGURE 7.1. Workflow for the "stack blocks 14, 8, and 11" command. The red arrows indicate a trace of execution of the workflow. Most resource demanding method is the "Plan input" STRIPS-like planner (circled red).

The main entry point in the workflow begins by receiving a stimulus of the form "(stack, 14, 8 ,11)". CRIS then proceeds to obtaining that stream from the I/O interface, placing it on the priority stack and performing a simple pre-processing, the result of which is to define that it is the textual input type that the system is dealing with and that it relates to the blocks world. It is then determined by the system that this request has to do with an already existing context—the system is dealing with blocks that had been introduced to the system in the past. After updating

the Active Context Stack (ACS) with a corresponding frame, a self model state of the system is query is performed. The system then goes into the mode of understanding what the objective of the current request is. This happens to be an imperative objective, which the system knows how to deal with. An axiom check is performed, which in this case is to see whether any unphysical behavior is scheduled to occur on the top level (e.g. the final state of the goal would lead to some unphysical object distribution). Once the axiom check provide a go-ahead to continue the process, the planner is executed. Several plans are found, among which the most optimal (with least number of steps) is selected and the required action is executed in the I/O module (show the state change on the screen).

The planner module serves for figuring out a sequence of actions that are to be performed in order to satisfy a user's request. The first point of entry to the planner in the loop (omitting the system's initialization) occurs when the token understanding unit calls the planner execution method, which invokes the blackbox planner through a system call. Once the blackbox finishes execution, the control goes back to the token understanding modules, which calls the plan analysis method, which provides a sequence of required actions to be performed.

Upon invocation, the user is presented with a GUI and an initial state of of the world is displayed on the screen. The initial state is read from a blackbox fact file, provided by the user as input. An example of such graphical representation is shown in Fig. 7.2. The same figure demonstrates the final stage of executing of a '(put,14,8,11)' command.

One of the two planners used to support the W0 execution was an off-the-shelf planner called "blackbox" [**98**]. Blackbox is a planning system converting STRIPS notation into Boolean satisfiability problems, and then solving the problems with a variety of engines. There is a great flexibility of engines to be used here. Among mentioned engines are graphplan [**19**], walksat [**169**], and satz [**119**]. The proposed engines can run for some time interchangeably, giving blackbox a possibility of functioning efficiently over a large range of problems. The second off-the-shelf planner, which was experimented with and used in order to collect the resource requirements results for the Blocks world is named "Madagascar" [**163**]. The Madagascar planner is an implementation of the SAT based techniques for planning.

W0 implements partial functionality of the original Winograd's SHRDLU system. Among the most important implemented features are the graphical world representation output, planner

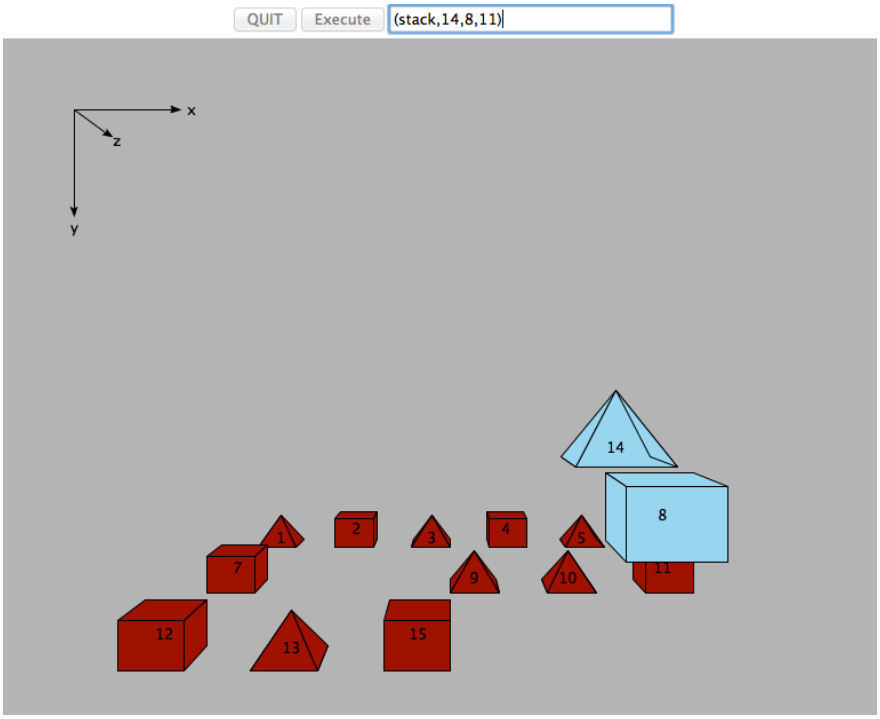

FIGURE 7.2. W0 world graphical representation at the end of executing (stack,14,8,11).

capable to provide a list of actions required to perform a request, as well as the closed system loop that provides interactiveness.

The scenario of driver application mapped on the workflow was implemented using programming language C++. The following hardware was used in order to obtain estimates on Floating Point Operations per Second (FLOPS) and memory capacity requirements of this driver application. All of the experimental data was obtained on a single Cray XE6 CPU-only compute node, with each node consisting of two 2.5 GHz AMD sockets with 8 cores per socket, and 2 threads per core, with a total of 32 processing cores. Each node had 6144KB, 2048KB, and 64KB of L3, L2, and L1 cache respectively and 64 GB of main memory. For every experimental run only one core was utilized, but the entire node with all 32 cores was exclusively requested. Measurements of FLOPS, main memory, and all related metrics were obtained using the CrayPat performance analysis tool [**97**].

Although the blocks world was implemented in the 1970s, the work on the planning component of the blocks world is still ongoing and has been active in the past few years in venues such as AIPS planning competition, and Intertnational Planning Competition [**11**, **51**, **87**].

In summary, a driver application named the blocks world that is mapped onto the generic workflow for MI was described above. It was shown how this application fits into the workflow, with detailed description of functional elements that take part in the blocks world request processing. Experimental setup, including the implementation and hardware details was provided and the relevance of modern problems of planning in the domain of PDDL and blocks world was summarized. Next section introduces another driver application, the 3-D facial recognition.

7.2. 3-D Facial Recognition

Facial recognition has been one of the most important problems of Artificial Intelligence of the past several decades. With 2014 Compound Annual Growth Rate (CAGR) of 9.5% and Total Addressable Market (TAM) of $1,307 Mn, this industry is expecting to continue its development with predicted TAM of $2,671.8 Mn in 2022 [**52**].

Facial recognition finds itself as one of the Symbolic Methods of CRIS abstract architecture. It is executed in "16: Execute planning/learning/recognition" step of the generic CRIS workflow (see Figure 7.3). Upon a request to the system to recognize a person in the photo or video stream, this request is processed and an objective that requires facial recognition utility is placed on the Active Objective Function Stack (AOS). Once that frame is popped, a process of achieving that objective begins by executing the workflow iteration. During that iteration, among other steps, pre-processing is performed that may include a simplified light approach of facial recognition (e.g. compare main facial features against existing database in a quick manner to see if a match can established). After a few additional steps, in case pre-processing could not achieve the goal, the main step of facial recognition is performed. Therefore, estimating the resource requirements for the facial recognition workflow in CRIS is equivalent to estimating the "Execute planning/learning/recognition" step. The pre-processing step, although is important for the workflow is execution, is not considered from resource requirements prospective, as it is negligibly smaller as compared to the 3-D facial recognition. The description of methodology for that step follows.

Some examples of applications of facial recognition include: i) web applications / social media or interaction for picture tagging, ii) surveillance, including security surveillance and surveillance to monitor customer behavior iii) identification, iv) authentication, v) livestock, animal, and pet

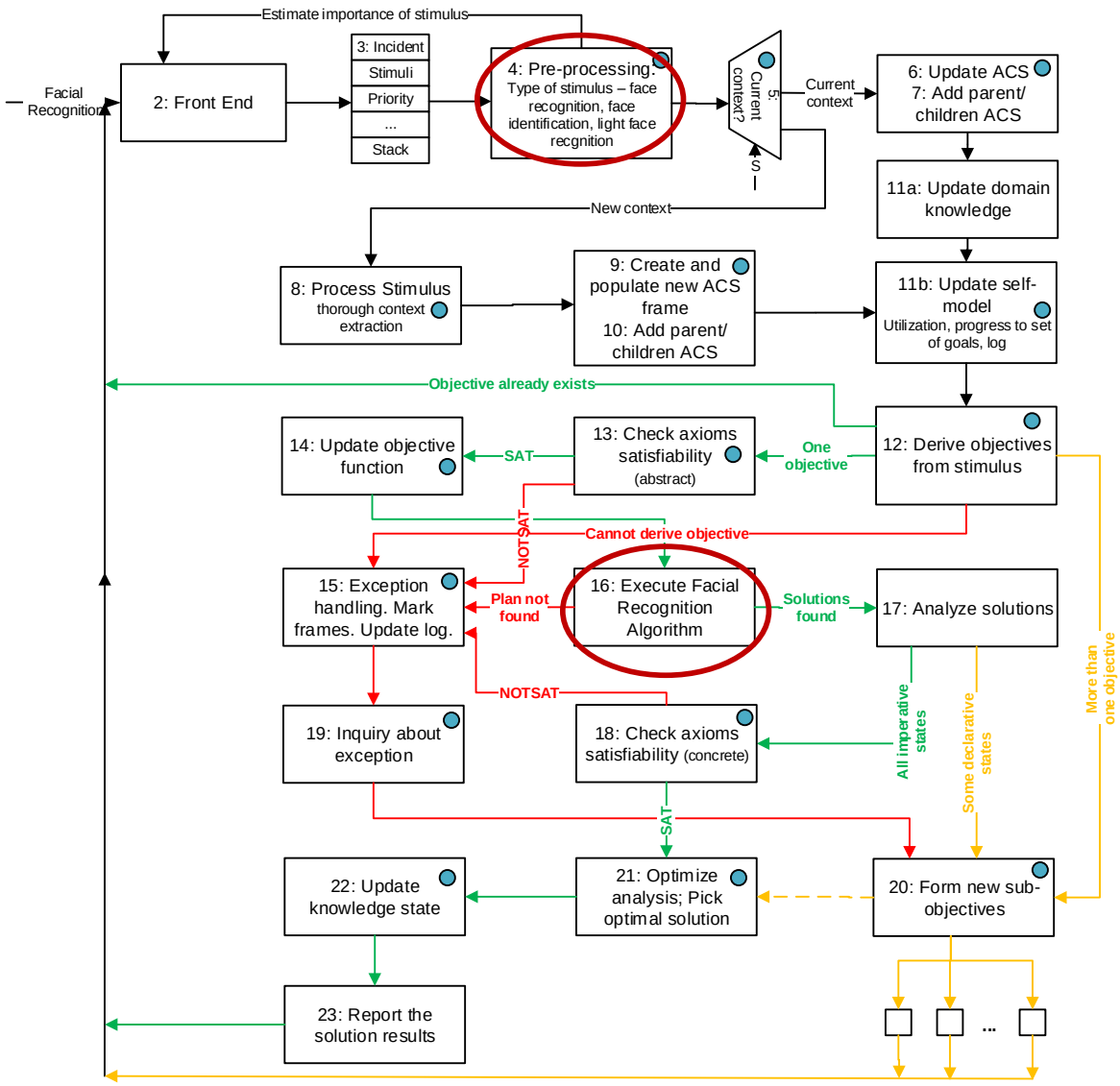

FIGURE 7.3. Facial recognition driver mapped onto generic CRIS workflow.

recognition, vi) body worn camera recognition systems, vii) security surveillance facial analytics, viii) device verification, ix) payment verification.

Two key factors in this area are the verification accuracy and computational costs associated with training and verification, with a desire to have high verification accuracy and operate in real-time (sub-two-second regime). Depending on the actual application, an ability to perform identification or recognition accurately from a video stream or an image in real-time may be critical, e.g. in areas such as airport/border/restricted access building security. The verification accuracy

is an important factor for any type of the recognition system, including social networks, security, surveillance, payment verification, etc.

Academic facial recognition research uses a few different classes of images in order to generate training models. The most common types of datasets are:

Visa images: These images originate from more than 100 countries, with generally an excellent pose and are in conformance with the ISO/IEC 19794-5 Full Frontal image type, which is an International Standard of Biometric Data Interchange Formats, described by National Institute of Standards and Technology (NIST). The images are of size 252×300 pixels with the mean interocular distance (IOD) of 69 pixels. Many images are live capture.

Selfie images: The selfie images vary in quality, while the portrait images are in reasonable conformance with the ISO/IEC 19794-5 Full Frontal image type. The images have mean IOD of 140 pixels. All images originate from the United States. All are live capture.

Webcam images: All portrait images, as in previous case, are in reasonable conformance with the ISO/IEC 19794-5 Full Frontal image type. The images have mean IOD of 68 pixels, they are of adult males from the United States. They all are live capture.

Non-cooperative images: Resolution varies widely, with a lot of images unconstrained, including a wide pose variation. Faces can be occluded. The images are of adults, all of them are live capture.

The application areas of algorithms that are working with these types of images vary. So, for example, visa images are used for border control, selfie and webcam images—for verification. In total, there are about 50 companies working in the facial recognition field. The false non-match rate (false negatives), or the probability that the recognition system incorrectly does not detect a match when it should, averages above 65% among the 53 commercial algorithms explored when using non-cooperative images. Such low rates can be explained by the nature of non-cooperative images dataset that is used for training. Occlusion of faces, variation in pose and resolution all affect the verification accuracy. Another important factor for this application of facial recognition that affects the verification accuracy is that the subjects do not have intentions to be recognized, as compared to applications of facial recognition on mobile devices or social media, where cooperation of subjects is implied. Additionally, some of such systems report even higher percentages of false positives (false match rates). High false non-match rates (FNMR) (65% average across 53

commercial algorithms with the best FNMR of 27.1% [**77**]) suggests that a shift from conventional Machine Learning (ML)-based 2-D approaches towards 3-D is required. The 2-D approach proves to have limitations when using certain types of train/test data, e.g. non-cooperative images. One alternative option might be to use an ML-based 3-D approach. This is not viable in the near term, however, since there is not a big enough database of 3-D facial point clouds to train an ML model.

A patent pending approach in 3-D Facial Recognition [**5**] was used in order to collect the performance data and estimate the resource requirements based on this application of MI. The innovation of the approach chosen is in optimal compression algorithm, which converts a collection of images that are acquired simultaneously from different angles into a 3-D facial point cloud, which in turn is transformed into an implicit surface function, which is on the order of 10–100 Kilobytes in size and is continuous. Such characteristics allow for fast and accurate biometric comparisons, including ear and nose features. The present approach varies from conventional commercial facial recognition software in the following aspects: i) it does not require ML matching, ii) it is 3-D based, iii) it allows to store a database of approx. one billion faces on a single Terabyte drive.

The fact that ML is not required alleviates the heavy computational costs that are normally required for the training step of ML algorithms used for facial recognition.

A description of the experimental setup for the Facial Recognition driver application follows. The system is presented with a static setup of a grid of 20 cameras shown in Figure 7.4. This is a homogeneous setup, i.e. all the cameras used for experiments were identical with the following specifications: Canon EOS 40D with 10.1 Megapixels digital SLR cameras. All of them were equipped with EF 50mm 1:18 II lenses. The shutter speed was programmatically set for 1/6 of a second, while the ISO speed was set to "auto".

There are two main steps involved into the definition of analytic functions that are used for matching against images from the existing database:

1) Pre-processing. This step consists of collecting multiple 2-D photographs from various angles of the camera grid. A few sample photographs that were collected from the grid's camera are presented in Figure 7.5. Once multiple photographs are acquired, a generation of 3-D point clouds by aligning those images occurs. The resulting 3-D point cloud is presented in Figure 7.6.

FIGURE 7.4. Camera grid setup of $4 \times 5 = 20$ cameras.

2) Representation of the 3-D analytic function by means of Radial Basis Functions (RBF) **[50]**. This step involves a number of small-to-medium intensity computations, which provide normal definitions at each point of the input point cloud, matrix and RHS assembly **[5,32]**. The objective of this approach is, given a set of distinct nodes $X = \{x_i\}_{i=1}^{N} \subset \mathbb{R}^3$, to define an interpolant $s : \mathbb{R}^3 \to \mathbb{R}$ of the form:

$$s(x) = p(x) + \sum_{i=1}^{N} \lambda_i \phi(|x - x_i|), \tag{7.1}$$

where p is a linear polynomial, the coefficients λ_i are real numbers and $|\cdot|$ is the Euclidean norm on $\mathbb{R}^3$. In this present implementation biharmonic RBFs are considered, therefore ϕ is chosen as follows $\phi(r) = r$. Consider the formulas below that define a matrix and RHS of the system of linear equations to be solved. Let $\{p_1, \ldots, p_l\}$ be a basis for polynomials of degree at most m and $c = (c_1, \ldots, c_l)$ be the coefficients that define p in terms of this basis.

FIGURE 7.5. Multiple 2-D images taken from different angles by the camera grid.

The matrix form of the equations to obtain the interpolant is then:

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = B \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}, \tag{7.2}$$

where

$$A_{i,j} = \phi(|x_i - x_j|), \quad i,j = 1,\ldots,N$$
$$P_{i,j} = p_j(x_i), \qquad i = 1,\ldots,N, \quad j = 1,\ldots,l.$$

If it is assumed that the polynomial part of the RBF in (Equation 7.1) is $p(x) = c_1 + c_2 x + c_3 y + c_4 z$, then $A_{i,j} = |x_i - x_j|, i,j = 1\ldots,N$, P is the matrix with ith row $(1,x_i,y_i,z_i), \lambda = (\lambda_1,\ldots,\lambda_N)^T$, and $c = (c_1,c_2,c_3,c_4)^T$. Now, solving the system (7.2) determines λ, and c, and hence $s(x)$.

Once the solution is generated, the interpolant $s(x)$ can be used to recreate a 3-D surface that represents a model of a face. A library named Scalable Linear Algebra PACKage (ScaLAPACK) [**18, 38**] was used in order to obtain the solution of equations (7.2). ScaLAPACK is a library of high-performance computing algebra routines for parallel distributed memory computations. It allows to achieve scalability, efficiency, and reliability due to

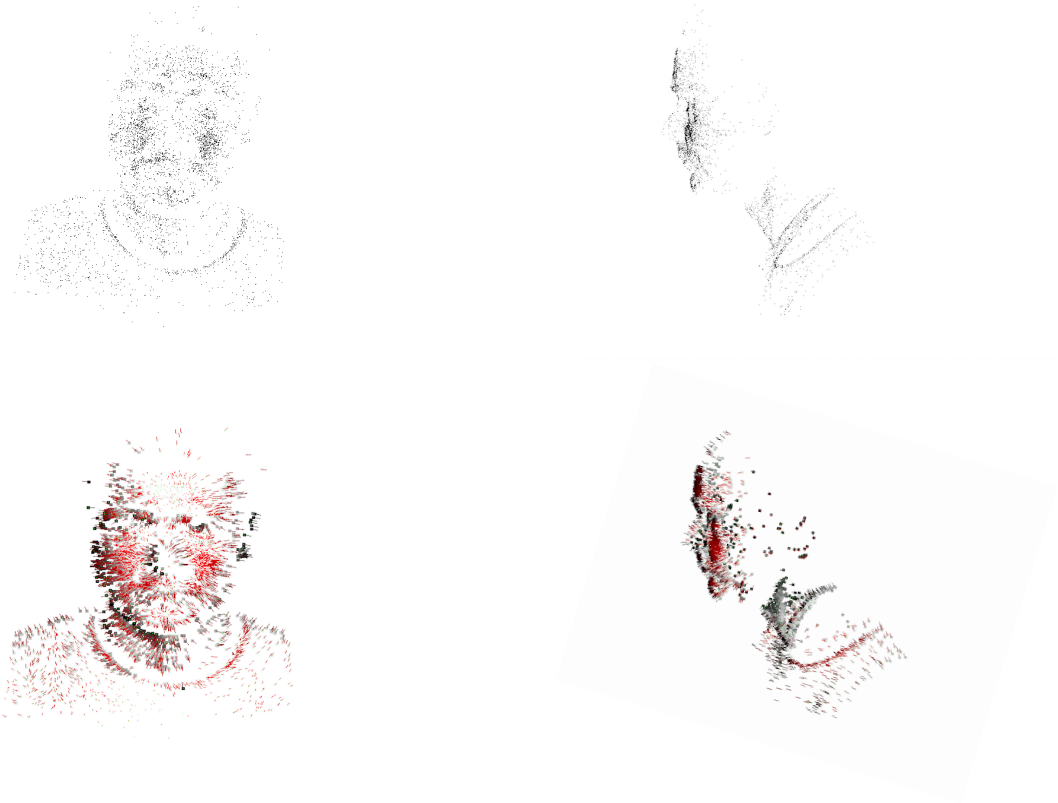

FIGURE 7.6. Point clouds (top) with normals pointed outward the surface at each point (bottom). Outward unit normals are represented by red arrows.

developing and promoting standards for low-level communication and computation routines. ScaLAPACK uses BLAS, LAPACK, and BLACS libraries.

The following hardware setup was used in order to perform the first step. The camera grid was combined into a hierarchy of USB hubs that were connected to one machine. The photographs were captured simultaneously by software that was run on that machine. The same machine was then used to perform the 3-D point cloud alignments. The pre-processing step, as well as normals definition and matrix/vector assembly, is negligibly small in terms of execution wall time and FLOPS, as compared to solving the system of resulting linear equations for the RBF interpolation. For comparison purposes, the execution times of the pre-processing step of generating 3-D point clouds are documented in Table 7.1.

The hardware specifications of a machine used for pre-processing are as follows: 2.5 GHz Intel Core i7-4870HQ processor with 4 cores, 6MB of L3 cache, 256KB of L2, and 32KB of L1 cache, with 16 GB of main memory.

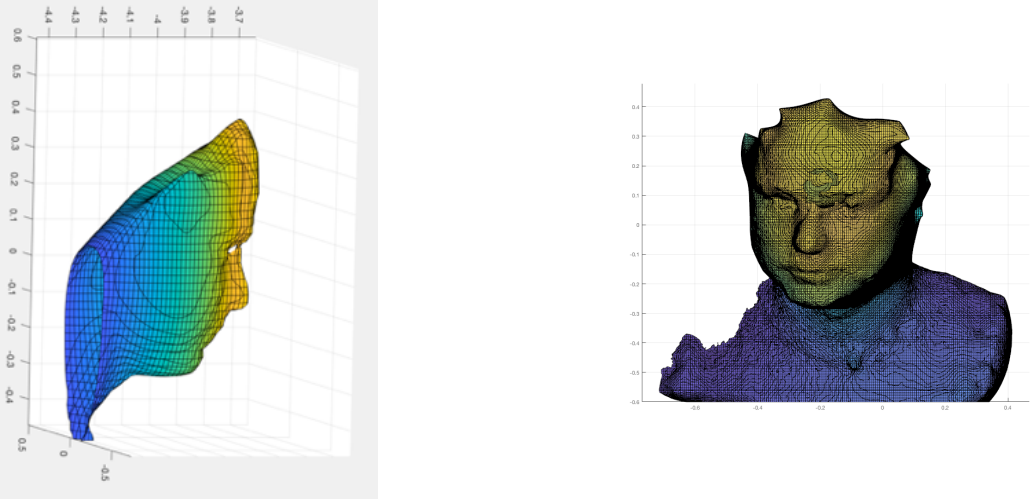

FIGURE 7.7. Analytic function 3-D surface plot. Low resolution on the left, high resolution—on the right.

Number of Cameras	Number of points	MPixels, aggregate	Wall time, s
5	1448	50.5	14
10	2529	101.0	29
15	4153	151.5	49
19	4985	191.9	62

TABLE 7.1. Numbers of cameras, corresponding points of 3-D point clouds, aggregate megapixels, and wall times required to obtain the point clouds.

The solution of the dense linear algebra problem was performed on the following hardware. All of the experimental data was obtained on a Cray XE6 CPU-only compute nodes, with each node consisting of two 2.5 GHz AMD sockets with 8 cores per socket, and 2 threads per core, with a total of 32 processing cores. Each node had 6144KB, 2048KB, and 64KB of L3, L2, and L1 cache respectively and 64 GB of main memory. The experiments were carried out using a range of $1,\ldots,1024$ compute cores, ranging from 1 to 32 compute nodes. For experiments that required less than 32 cores, an exclusive access of the whole node was requested. Measurements of FLOPS, main memory, and all related metrics were obtained using the CrayPat performance analysis tool [**97**]. The approach of finding an interpolant requires a system of linear equations with surface points, as well as outward normals defined in each point. The latter are shown on

the bottom of Figure 7.6. The result of the implicit surface interpolation, that was built based on the obtained solution, is shown in Figure 7.7.

The computational costs associated with the RBF interpolation, which is the most expensive step of the considered 3-D facial recognition approach, can be estimated analytically using the following formula [**32**]:

$$\text{Ops} = N^3/6 + \mathcal{O}(N^2) \tag{7.3}$$

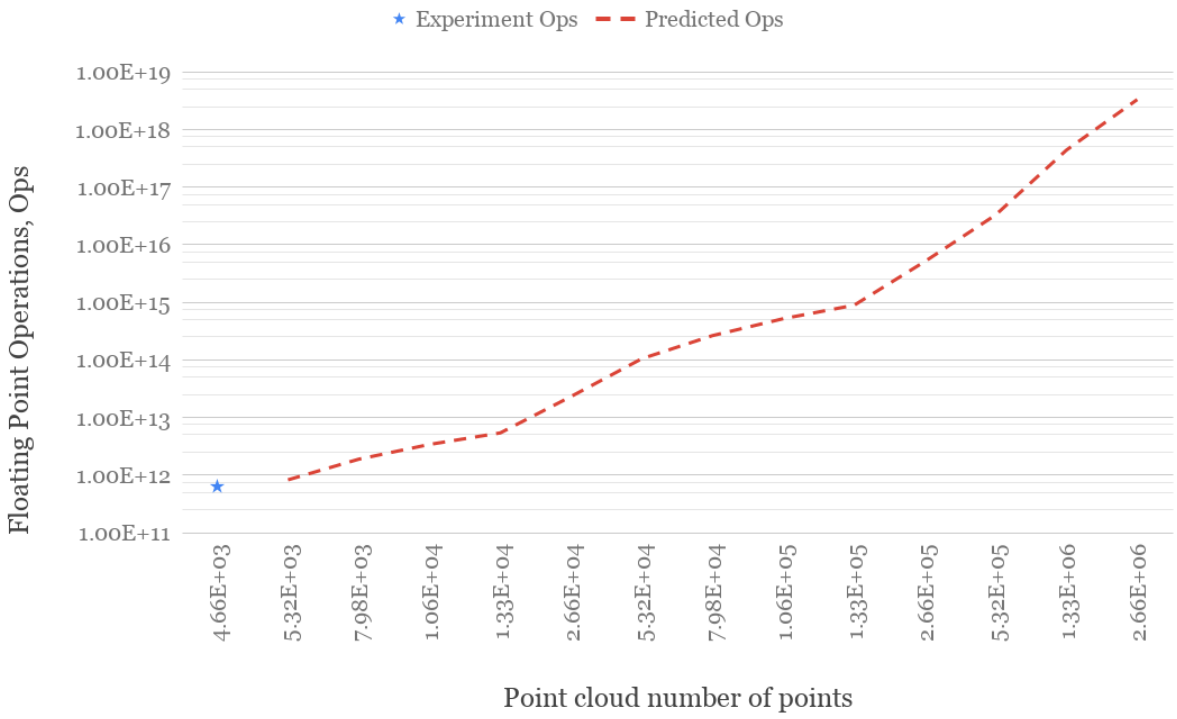

FIGURE 7.8. Predicted Ops vs number of points for RBF interpolation. The point cloud of 19 cameras (4663 points) is predicted to incur approximately 550 GOps. One POps is expected to occur at approximately 150K points.

Based on this formula, an estimate of the amount of Ops achieved for various number of points is presented in Figure 7.8. The estimates predict that for $N = 4663$ the number of Ops $\approx 20 \times 10^9$, which is consistent with the amount of actual Ops obtained by running this experiment. The two estimates are discussed in the next chapter.

7.3. Autonomous Moving Agent

The Autonomous Moving Agent is the third and last driver application mapped onto the workflow of a system for MI. Very much like the 3-D facial recognition, measurements of a particular functional element of the workflow is considered for resource requirements analysis.

A iRobot Roomba 650 vacuum cleaner was used as a platform to support the AMA. The normal behavior of internal roomba's implementation is overridden by using a control computer Tegra TK1 by NVIDIA (see Figure 7.9).

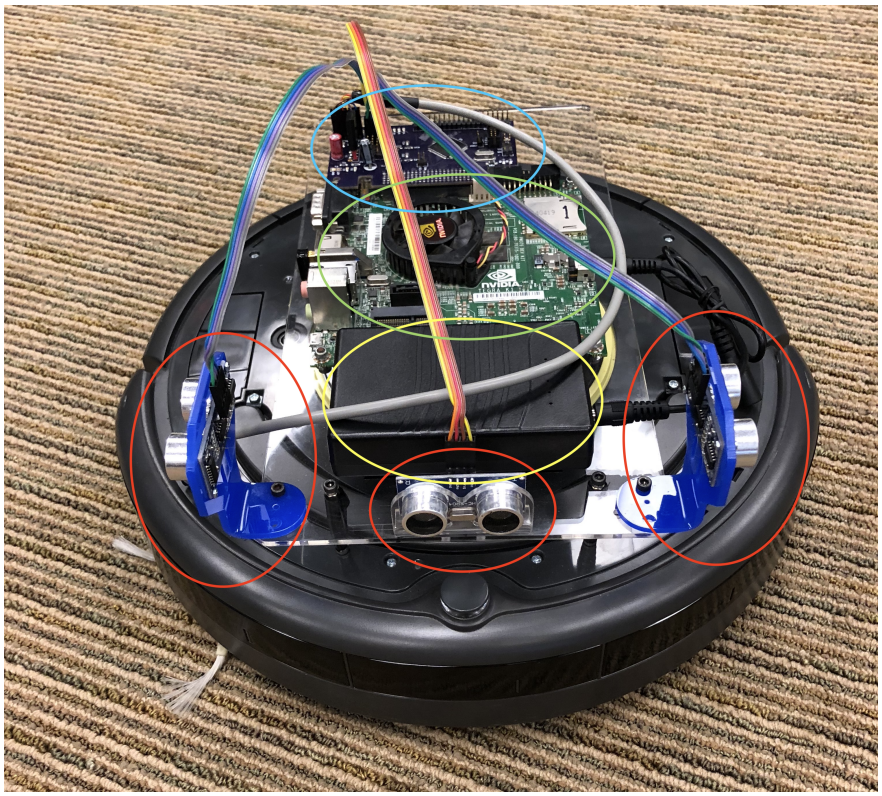

FIGURE 7.9. The AMA hardware setup. The entire system is physically hosted by a roomba vacuum cleaner. A controller, the NVIDIA Tegra TK1 Jetson board, is circled green, the custom-design breakout board is circled sky-blue, the ultrasonic distance sensors are circled red, and the external power supply is circled yellow.

The external board used as a controller is NVIDIA Jetson TK1 based on embedded NVIDIA quad-core CPU ARM Cortex-A15, which is a 32 bit processor. The board is also equipped with an NVIDIA Kepler GPU with 192 CUDA cores and has the following specs: 2 GB main memory, 16 GB 4.51 eMMC, 1 half mini-PCIE slot, 1 SD/MMC connector, 1 full size HDMI port, 1 USB 2.0 port, 1 USD 3.0 port, 1 RS232 serial port, 1 ALC5639 Realtek audio codec, 1 RTL8111GS Realtek

GigE LAN, 1 SATA data port, and SPI 4 MByte boot flash. The following signals are supported through an expansion port: DP/LVDS, SPI, GPIOs, UART, HSIC, I2C [**149**].

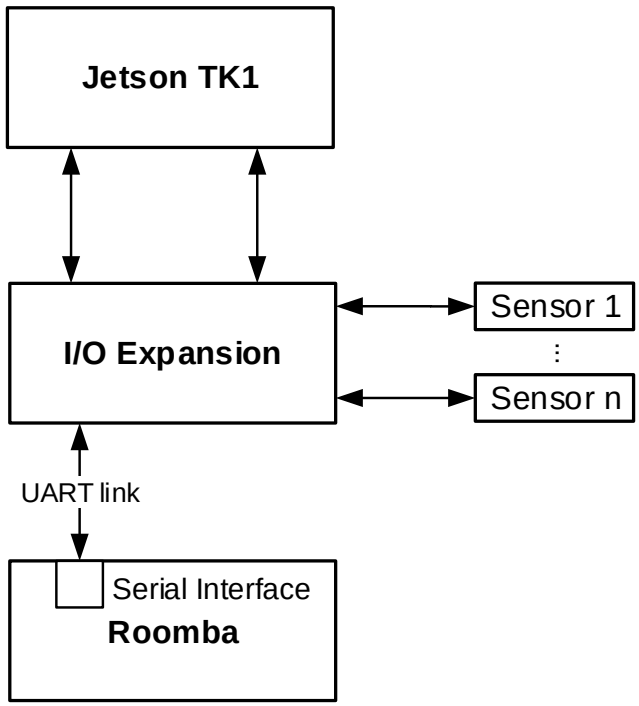

FIGURE 7.10. Schematic representation of the autonomous moving vehicle architecture. Roomba is a physical host for the entire setup. It is communicating with the I/O expansion board via a UART link. The expansion board is communicating to distance sensors (ultrasound, laser) connected to it, as well as to the Jetson TK1 controller board via the SPI and i2c interfaces.

It is not possible to interface the TK1 directly with the Roomba, so an additional custom-design board was introduced into the setup (see Figure 7.10). A custom-design breakout board, or expansion board, is used in order to address the interfacing between the roomba and TK1. Additionally, the breakout board provides better precision for I/O timing and data acquisition. Moreover, the breakout board is necessary to maintain voltage level translation and provide communication over SPI bus, and I2c bus.

Two types of sensors were used for this experiment. The first hardware revision included ultrasonic sensor, HR-SR04 by Smraza. The basic operation principle for that sensors is as follows: ranging is triggered from the sensor, at which point eight 40kHz square waves are emitted

automatically, which is followed be the sensor tests to determine whether there is any signal returned. If there is signal returned, the high level signal is the time that was taken for the signal to travel and return back. Therefore, $S = T \times c/2$, where S is the testing distance, T — duration of high level signal, $c = 340m/s$ — speed of sound. After some initial tests with these sensors it was discovered that the precision was not satisfactory for the project, therefore the laser sensors were used instead. The second type of sensors, which was used to perform the experiments was ST VL53L0X [**188**]. It is a Time-of-Flight laser-ranging module that provides accurate distance measurement. The VL53L0X's 940 nm Vertical Cavity Surface-Emitting Laser (VCSEL) emitter is totally invisible to the human eye, and enables long ranging distances, high immunity to ambient light, and better robustness. The VL53L0X performed more robust, and was a better fit for this application, as compared to the previously used ultrasonic sensor.

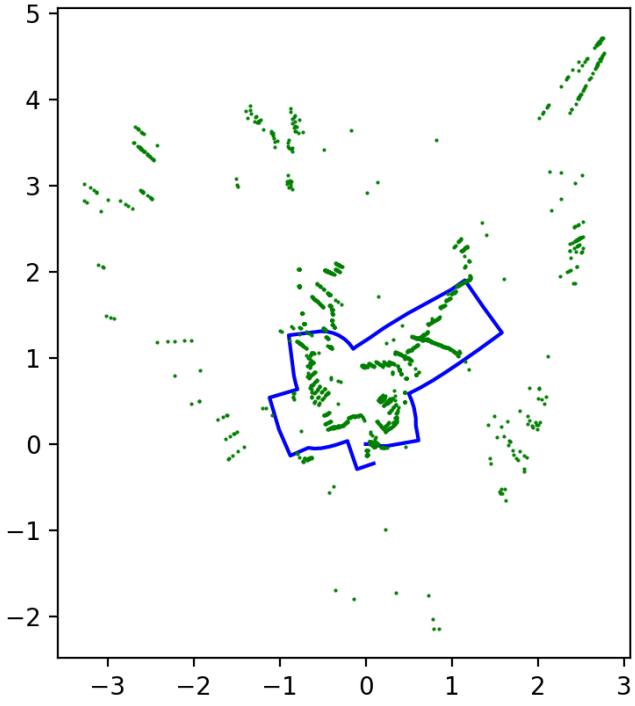

FIGURE 7.11. AMA's trajectory of the wall follower example. One lap of following the wall was documented in this experiment. The blue trajectory line shows the path, which the AMA was generating during the navigation. The green dots represent physical obstacles that were documented during the navigation.

An algorithm used to obtain the resource requirements is autonomous wall follower agent. The objective of the AMA in this scenario is to follow a wall indefinitely, based on the sensor data constantly maintained throughout the execution. While following the wall, the AMA is constantly monitoring for changes in the surrounding environment via the three laser sensors. Whenever a wall that is followed ends, the algorithm reevaluates the surroundings and makes necessary adjustments in the travel trajectory. A sample run of eight laps of the AMA for the wall follow scenario is shown in Figure 7.11. The blue lines correspond to the route that the agent had traveled, while the green dots are the obstacles that were identified by the agent's sensors along the way. The operations associated with these computations are measured and presented in the next chapter.

7.4. Summary

This chapter serves an important purpose of connecting the existing architecture and workflow for MI with practical aspects by presenting how external drivers fit onto the generic workflow. Next, this chapter provides details of the methodology and experimental setup that has been undertaken to estimate resource requirements. For one external driver, the 3-D facial recognition, expected requirements based on analytic estimation functions for Ops and amount of memory are provided. These estimates allow to judge about the order of resource requirements imposed on hardware for these applications to perform in real time. These theoretical results are further used to compare against in the next chapter, which presents the experimental results of the external drive for MI.

8

Experimental results

This chapter serves as a report of the actual experimental results obtained by subjecting the external drivers onto the Cognitive Real-time Interactive System (CRIS) workflow. These experiments were a foundation for obtaining results that are used in the next chapter to draw conclusions on the characteristics of hardware resource requirements necessary to support various kinds of applications of Machine Intelligence (MI). The experimental data for all three external drivers are reported and analyzed in the following sections.

8.1. Blocks World

This Section reports the results of the experiments that were obtained by subjecting an external driver application of the Blocks world onto the CRIS workflow. All the experiments were executed with variable number of blocks world sizes, varying from 4 to 28. All of these results were obtained with a planner named Madagascar, which is an implementation of the SAT based techniques for planning [**123**, **162**]. The input data for variable blocks world sizes and tasks were taken from the AIPS-00 planning competition [**2**]. The results of the performance and the amount of work that were achieved are presented in Figure 8.1. The amount of work (Ops) grows monotonically and observes two orders of magnitude change, while the average achieved performance (FLOPS) is decreasing as the blocks world is growing in size. A few non-monotonic fluctuations that occur in the observed performance graph are common in real-time systems.

The memory resource requirements for the Blocks world external driver are presented in Figure 8.2. The amount of main memory that is consumed by this external driver is limited by approximately 3.5 GB. This much memory is required when operating on a world of 28 blocks. The memory requirements are monotonically distributed as the number of blocks is increasing.

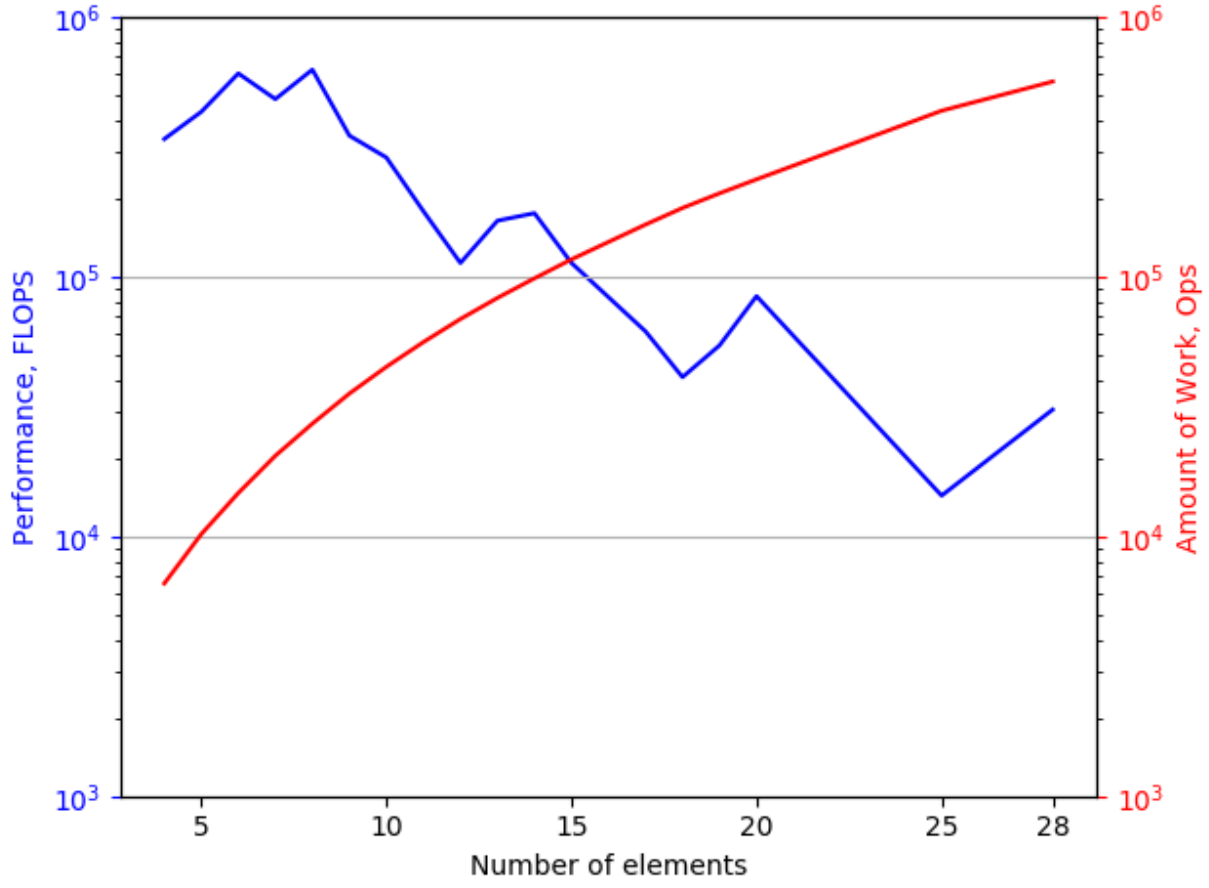

FIGURE 8.1. Number of Ops and FLOPS observed when running CRIS with blocks world as external driver. The Ops are monotonically increasing, as expected due to more computational requirements proportional to number of blocks. The FLOPS are decreasing with the increase in number of blocks.

One limitation of the planner implementation (Madagascar) that was used to obtain the results presented above is that it does not behave well on the number of blocks beyond 20. Although various reports suggest that this implementation, along with other best performing in AIPS-00, including System R, LPG-td, and Fast-Forward (FF) **[2]** were able to provide results for up to a few hundreds of blocks, this behavior could not be replicated on the architecture that was used to conduct the Blocks world experiments on a Cray XE6 CPU compute node, with each node consisting of two 2.5 GHz AMD sockets with 8 cores per socket, and 2 threads per core, with a total of 32 processing cores and 64 GB of main memory. The System R code was not publicly available. The LPG-td version was somewhat outdated (latest version available was released in September 2003) and the source code could not be fully compiled on BigRed2 due to issues with flex and bison

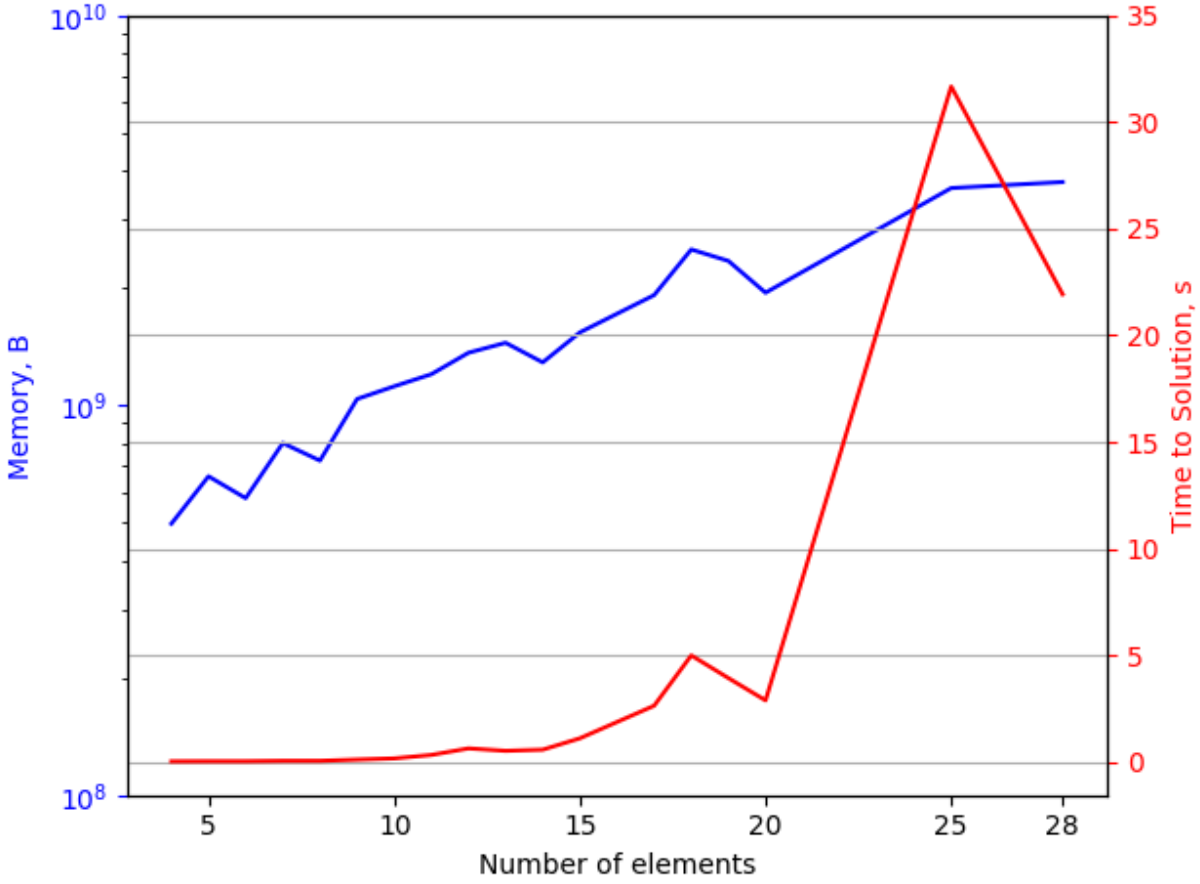

FIGURE 8.2. Amount of main memory, B and time to solution, s vs number of blocks. The main memory consumption grows proportionally to the number of blocks.

generators. The implementation of FF was available and compiled, however the amount of FLOPS was too small in order to include in the thesis. The execution times, though, were acceptable and for certain block numbers were faster than those of the Madagascar. Nevertheless, the main issue of FF was that according to the reports of AIPS-00 [**2**] and another study [**90**], the system was not stable and did not yield results for all number of blocks provided as input. That behavior was confirmed on BigRed2, where the FF failed to provide a result in about 34% of test cases. To sum it up, the Madagascar planner performed best among all of the other planner implementations named above.

The blocks world demonstrates a middle ground external driver among all drivers of CRIS considered in this work, where the resource requirements vary from tens to hundreds of thousand

of FLOPS with the real-time processing constraint satisfied. The amount of main memory that is required in order to support this class of drivers varies from hundreds of Megabytes to Gigabytes. The real-time processing requirement is only satisfied for smaller problem sizes (up to 15 blocks). Beyond that number, the execution time varies from seconds to tens of seconds.

8.2. 3-D Facial Recognition

The 3-D facial recognition involved a pipeline of two main steps: i) pre-processing, and ii) RBF interpolation. The resource requirements of the pre-processing step are negligibly small, as compared to the second step. They were briefly discussed in the previous chapter, see Table 7.1. The rest of section will be dedicated to reporting and discussing the results obtained by performing the RBF interpolation, more specifically to solving a system of dense linear equations using the library ScaLAPACK [**18**].

Figure 8.3 presents the time to solution graphs of the linear solver PDGESV [**156**], which is a direct solver that uses LU decomposition. All of the experiments described in this Section were performed with $n = 4,663$ points of the 3-D point cloud. The methodology that was used to obtain the solution [**5**] requires incorporating twice as many points (including the points that are unit normals to the original points), therefore the size of the matrix was $N = (4,663 \times 2)^2 = 86,974,276$. Various plots in the graph represent experiments that were run with variable matrix block size values, denoted NB. Based on the results in the Figure, the best block size value is NB=50 for number of cores equal to 128 and above and NB=40,50,100 for up to 128 cores. These results show that for a few values of NB execution time of less than one second is obtained on 128+ cores. The system is operating in real-time in this regime.

Figure 8.4 presents the results of Radial Basis Functions (RBF) interpolation utilizing the ScaLAPACK pdgesv function. The results yield the best speedup $S \approx 61.4$ when executed on 512 cores with the block size of $NB = 20$. Speedup of 40-60 is observed with block sizes of 10, 20, 30, and 40.

An important result, which provides a characteristic regarding the number of FLOPS achieved on a particular hardware system is provided in Figure 8.5. The blue line corresponds to the amount of achievable parallelism which is observed on 1..1024 cores with the max FLOPS number of 0.63 TFLOPS. The time to solution for this number of FLOPS was $t = 0.89$ seconds, which brings this

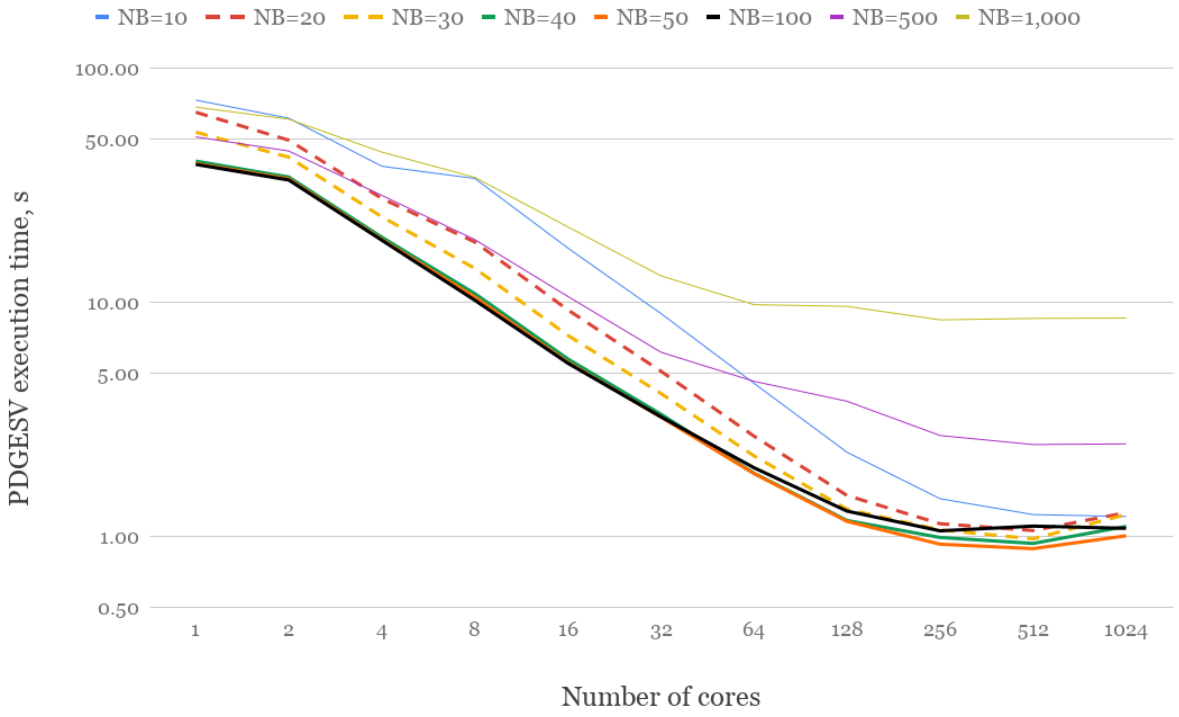

FIGURE 8.3. ScaLAPACK PDGESV wall time time for point cloud of 4,663 points. The best performing values of block size are NB $= 40, 50, 100$. The block size NB $= 50$ (orange solid line) is the fastest, executing in less than one second on 256+ cores.

computation to the class of real-time problems of MI. The red line signifies the maximum theoretical performance that is achievable on this hardware system. The fact that the blue and red lines are diverging can be explained by Figure 8.6, which is a ratio of achieved FLOPS and maximum theoretical peak FLOPS for this system. Such low value of the ratio suggests that the application of MI on the HPC systems is far less optimal than what is achieved with high-performance Linpack (HPL), a software package that solves a random dense linear system in double precision arithmetic on distributed-memory computers **[155]**. HPL is regarded a standard benchmark for ranking the world's fastest supercomputers on the Top500 list.

Finally, the main memory consumption is provided in Figure 8.7. The actual amount of used main memory is presented by a blue line, while the red line depicts the maximum available main memory for variable number of compute cores. The amount of main memory used in non-distributed regime was bounded by approximately 1.5GB, while for the largest number of parallel cores the amount of consumed memory reached approximately 43GB. Again, as in the case with

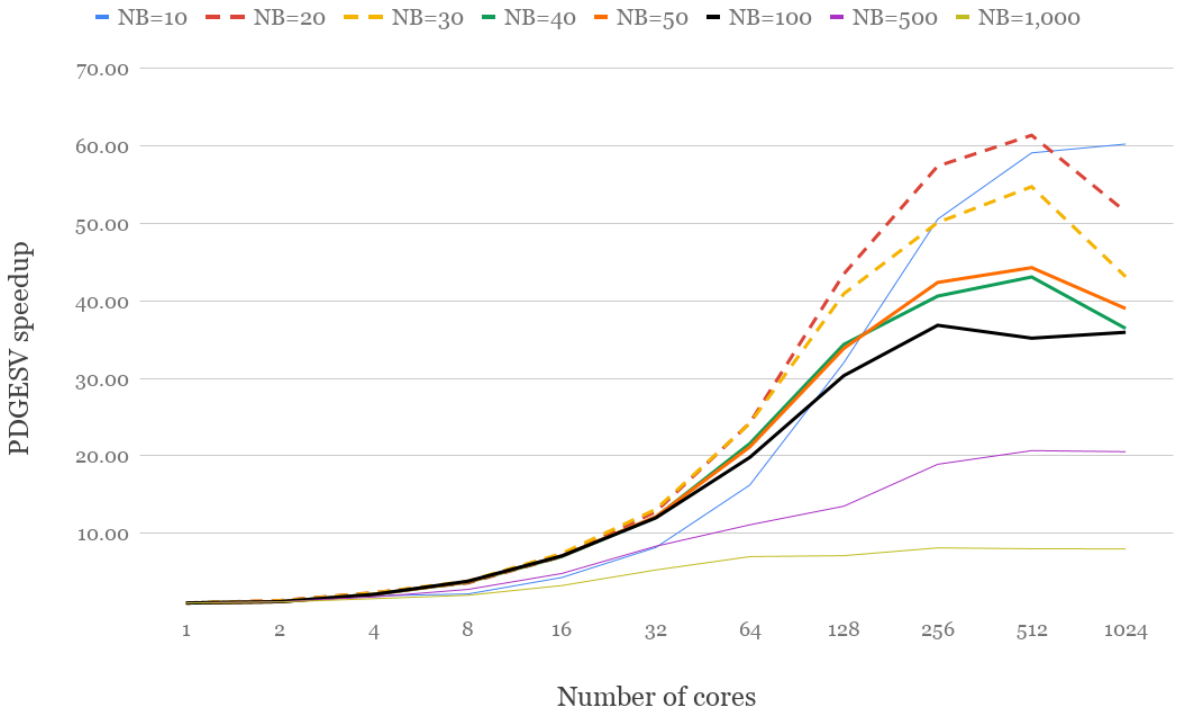

FIGURE 8.4. ScaLAPACK PDGESV speedup (T_N/T_1) for point cloud of 4,663 points. Line colors and shapes are kept the same as in Figure 8.3 for comparison purposes. Values of block size NB $= 10, 20, 30$ are best for speedup, with NB $= 20$ being the leader on 64..512 cores. Best speedup on 1024 cores is observed with NB $= 10$.

FLOPS, the percentage of actual memory used for this driver application of MI was really low, as compared to the amount of memory available on this particular system.

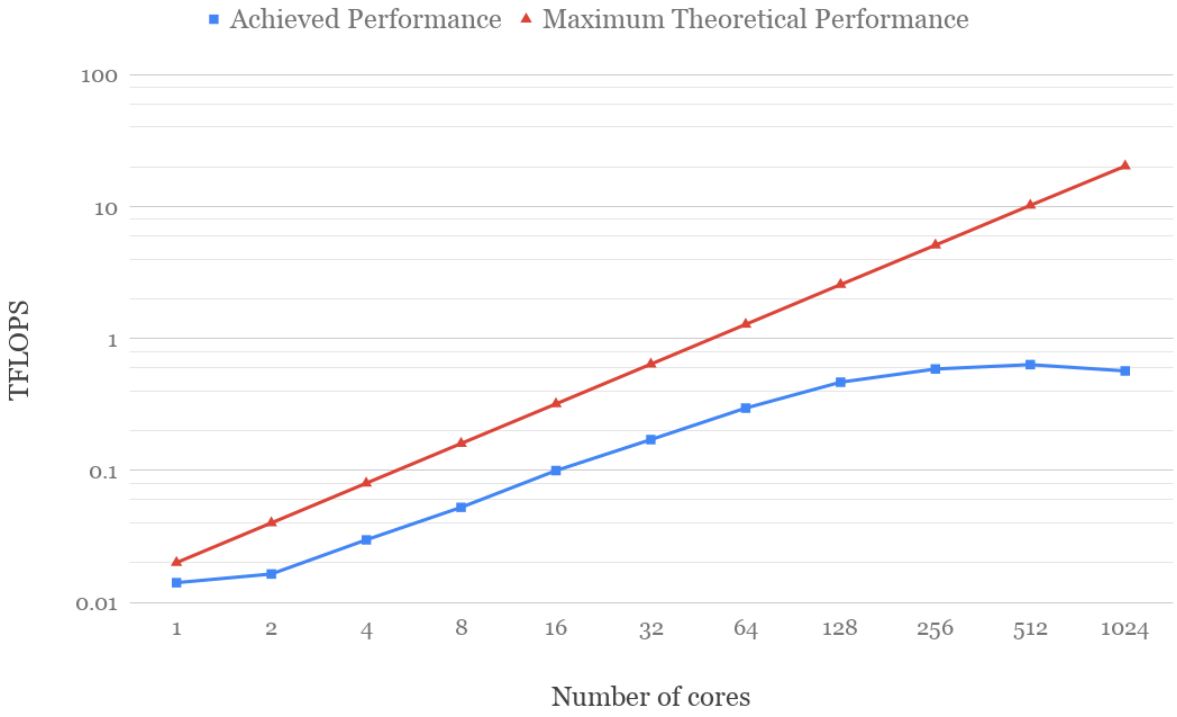

FIGURE 8.5. ScaLAPACK PDGESV floating point operations per second for point-cloud of 4,663 points, TFLOPS. The blue line shows the TFLOPS achieved on 1..1024 cores, with best value of approx. 0.63 TFLOPS on 512 cores. The value of observed FLOPS is 6.2% of the R_{peak}.

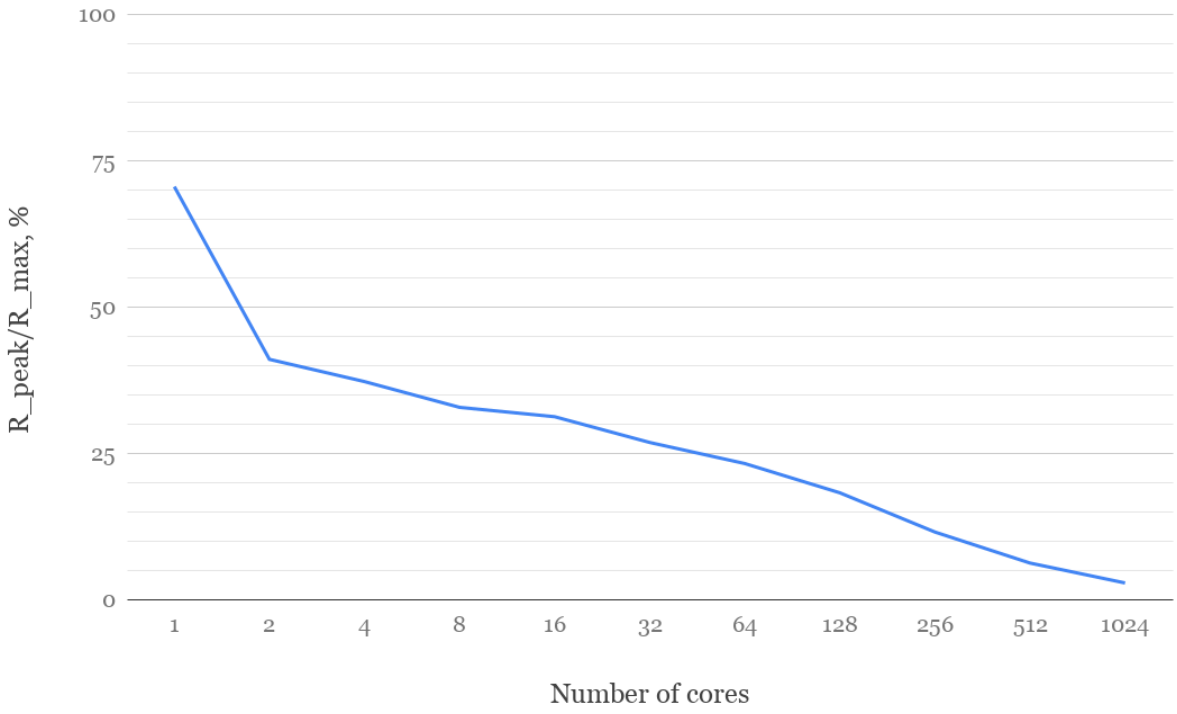

FIGURE 8.6. Ratio of achieved FLOPS and theoretical peak performance, R/R_{peak} for pointcloud of 4663 points, percent.

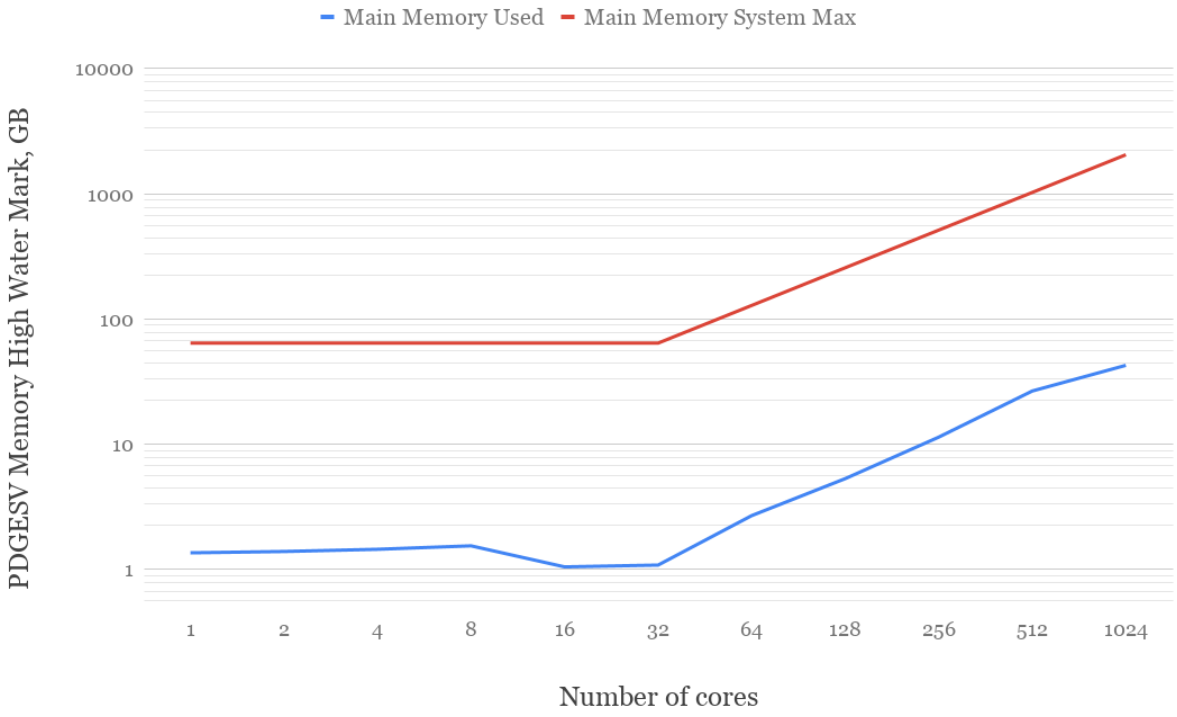

FIGURE 8.7. ScaLAPACK PDGESV memory high water mark for pointcloud of 4,663 points, GB=$10^9\times$ bytes. The main memory utilization stayed constant in non-distributed regime up to 32 cores at approx. 1–1.5GB, which was 1.63–2.40% of overall peak available memory. The maximum consumed memory was 42.53GB for the case of 1024 cores (64 nodes) , which was approx. 2.07% of overall available system memory.

8.3. Autonomous Moving Agent

The measurements obtained for the Autonomous Moving Agent (AMA) wall follow external driver were taken for the number of wall elements varying from one to seven. The AMA is programmed to follow the wall indefinitely. Figure 8.8 demonstrates the amount of work, measured in Ops performed, as well as the achieved performance, in FLOPS that were required in order to complete the task.

The autonomous moving agent performs a task of following a wall until given a command to stop. This set of experiments had the wall obstacle change size from one to seven. Figure 8.8 contains the results of this problem execution. The graphs represent the performance (blue), and amount of work (red) vs the number of wall elements. The results show that the amount of

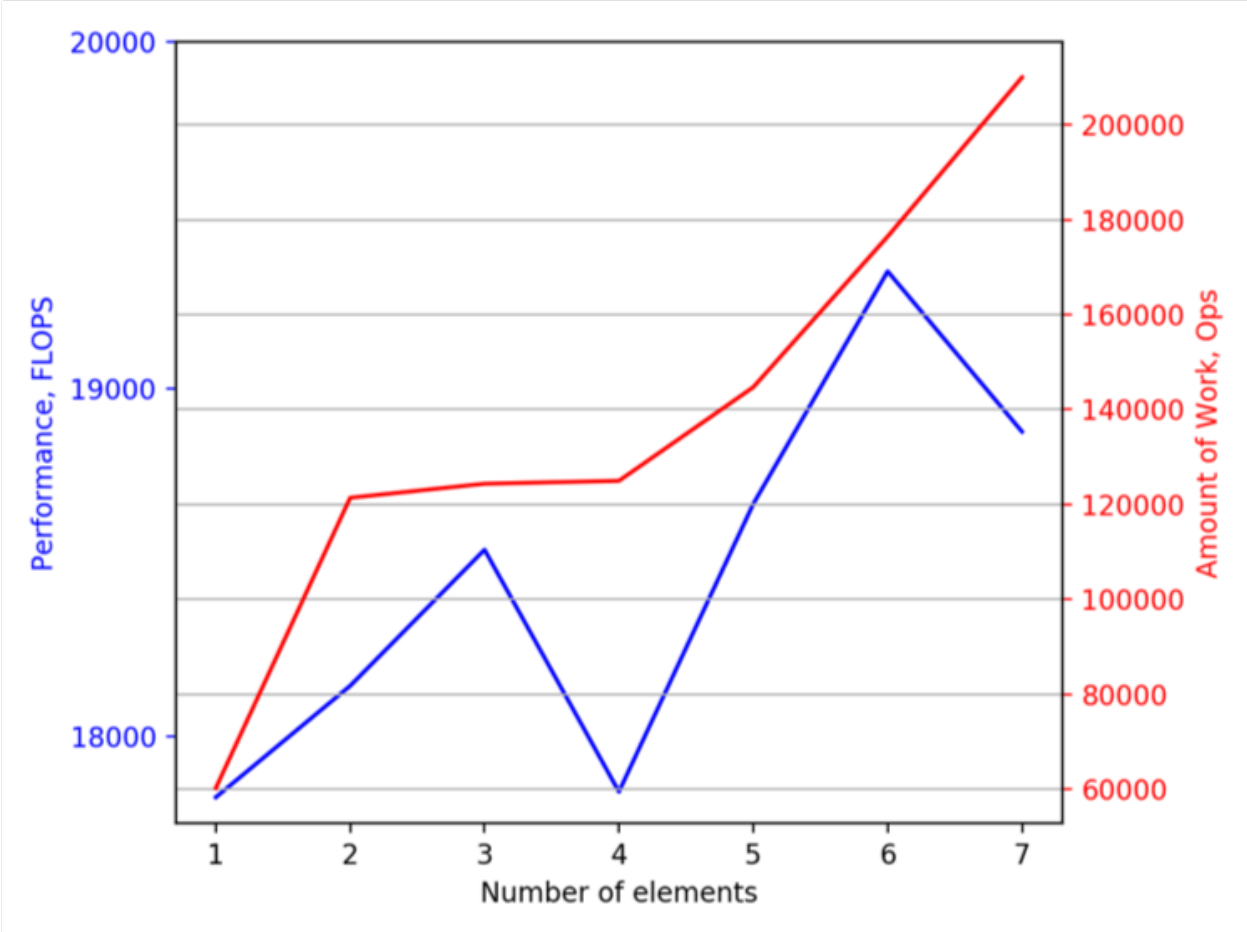

FIGURE 8.8. Performance, FLOPS and amount of work, Ops vs number of obstacles for the problem of AMA. The number of FLOPS achieved is on the order of ten thousand.

work, measured in Ops, is monotonically increasing as the number of wall elements is increasing. The performance observes some fluctuations with increase in the number of wall elements, however these fluctuations are minor with the biggest one being only four percent decrease from 18.5 KFLOPS to 17.8 KFLOPS, which was observed with wall sizes of three and four respectively.

The main memory consumption for the Roomba AMA are as follows: the size of the virtual memory (the total memory that is available to the application), VmSize, is 46,416 KB; the virtual memory resident set size, VmRSS, which is how much actual memory is consumed by the process is 3,936 KB. Both of the virtual memory values did not vary with varying the number of wall elements.

8.4. Summary

The results of three various external driver application for the CRIS workflow were presented in this Chapter. The results appear to be consistent and as expected as compared to the assumptions made about the drivers. Certain irregularities for some applications were discussed separately where applicable. Along with the FLOPS and main memory requirement reports provided, a few other observations were made. The three applications considered present a range of problems, varying in the amounts of FLOPS imposed on hardware systems in order to achieve real-time processing from tens of thousands of FLOPS ($\mathcal{O}(10^4)$ FLOPS) to under a TeraFLOPS ($\mathcal{O}(10^{11})$, with AMA being the least demanding (best achieved performance of 1.9×10^4 FLOPS), the blocks world planner in the middle (best achieved performance of 3.3×10^5 FLOPS), and the 3-D facial recognition being the most computationally expensive example of MI with the best achieved performance of 6.34×10^{11} FLOPS .

The next Chapter provides analysis of these results and provides the bounds on resource requirements for MI, as well as speculates what the future trends in MI and related High Performance Computing (HPC) architectures for MI may be.

9

Evaluation and Analysis

This chapter serves a dual purpose: i) it gives a summary of the results that show where the bounds on resource requirements for Machine Intelligence (MI) are today; ii) it provides estimates of the future technology trends, as well as the trends of MI and what the future development of technology means for the field of MI.

Due to rapid technology change in recent decades and some major changes in hardware architectures, powerful MI tools have been increasingly and successfully applied to areas of speech recognition, translation, image recognition, and many other tasks. A lot of such applications take advantage of specialized hardware, especially Graphics Processing Units (GPUs). These chips are usually used for most computationally expensive phase—training. With the increasing demand of software requirements for various MI problems, companies are competing in delivering new hardware that will allow even better performance improvements for such problems. Most efforts focus on accelerators that are specifically tailored for certain applications (e.g. deep neural networks). This class of accelerators is commonly referred to as Application-Specific Integrated Circuits (ASICs). Modern ASICs oftentimes comprise microprocessors, various flavors of memory (ROM, RAM, flash, etc.) and Field-Programmable Gate Arrays (FPGAs). Some examples of ASICs are Google's TensorFlow Processor Unit [**95**], Facebook's and Intel's collaboration on the Neural Network Processor Nervana [**75**], and NVIDIA's Deep Learning Accelerator architecture [**148**]. There is a clear development cycle for the systems of MI—under ASICs. With the current technology growth, it is expected that the amounts of input sensor data for MI applications will increase at rates faster than those for the hardware advancements. In spite of the current advancements with GPUs and special hardware ASICs, it is expected that the rate of growth of the current High Performance Computing (HPC) systems will not match the growth rate of MI input sensor data.

That may be a limiting factor for real-time applications of MI. It is clear, however, that Cognitive Real-time Interactive System (CRIS) fits into the technology development agenda under the development of ASICs. To give a concrete idea of how things are changing, a specific example of MI, 3-D facial recognition is considered in this chapter.

The overview of this chapter is as follows. A summary of resource requirements for today's applications of Machine Intelligence on an example of the 3-D facial recognition as an external driver is discussed in Section 9.1. This particular driver application was chosen due to it having the highest resource demands out of all applications considered (see results analysis in Chapter 8). Hence, analyzing this application provides the best lower bound estimate on resource requirements among all applications of MI considered in this work. As part of this exploration, it has been discovered that HPC is an important component in obtaining the solution of the 3-D facial recognition problem. A number of estimates on how the technology will be driving the development of HPC systems for MI is provided in Section 9.2. Finally, Section 9.3 provides a summary of the future trends analysis.

9.1. Current resource requirements for MI

The analysis of the current resource requirements for MI provided in this section is restricted to an example of 3-D facial recognition (see Figure 9.1).

The behavior of the achieved performance in FLOPS shows that there is approximately one order of magnitude of exploitable achieved performance for this experiment, however the asymptotic behavior of this function (i.e. the change of rate beyond 128 cores) suggests potential limitations with scalability on large number of cores. In addition, a change of the achieved performance growth rate as compared to the theoretical maximum implies a reduction in efficiency of computation, which will be discussed later in this chapter. This reduction suggests that there is a limitation on the exploited parallelism when executed on the current hardware.

The measurements are generated using a linear algebra library, ScaLAPACK, for solving a system of dense linear algebraic equations in order to obtain an analytic function that determines a person's 3-D facial representation. The blue line in the graph shows the actual performance gains that were achieved of a range of 1..1024 compute cores of a supercomputer BigRed II, while the red line shows the maximal theoretical performance for this hardware system.

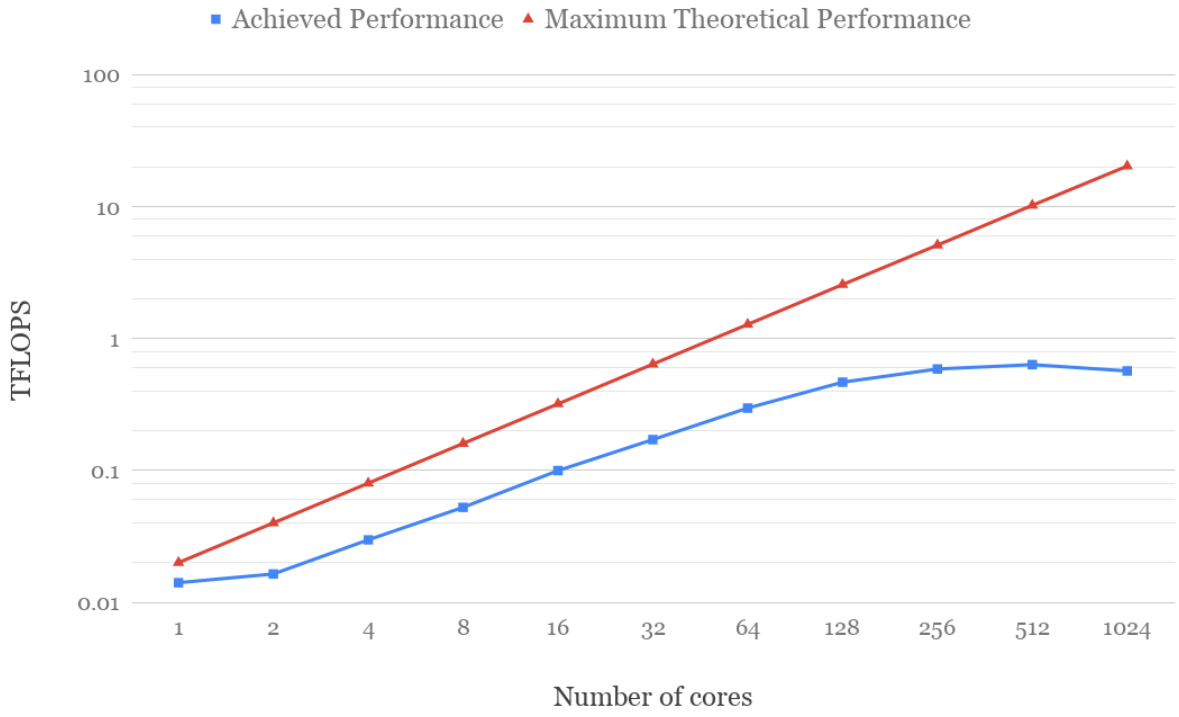

FIGURE 9.1. Number of FLOPS achieved on various number of compute cores. The best performance was achieved on 512 cores and is approximately $R = 0.63$ TFLOPS, which is 6.2% of the theoretical maximum $R_{peak} = 10.2$ TFLOPS.

The main outcome of the result presented in Figure 9.1, however, is that the lower bound resource requirements for the 3-D facial recognition problem, and hence for today's field of MI is approximately 0.63 TFLOPS, or 0.63×10^{12} FLOPS. This value is the point where the blue graph asymptotes, and it is actually the maximum achieved FLOPS for the current experiment. It is also important to note that when this problem is executed on 512 cores the time to solution value is approximately 0.87 seconds, which is in the range of real-time processing. The lowest number of cores that allows for the algorithm to achieve sub-second real-time execution is 256, with time to solution of approximately 0.93 seconds and the achieved performance of 0.59 TFLOPS. From all of the considerations above it follows that

> there exists an example of a problem of Machine Intelligence, a solution to which in real-time requires a hardware system with at least 0.63×10^{12} Floating Point Operations per Second of achieved performance.

There are not any examples among all the classes of problems of MI considered in this work that would exceed this requirement imposed on the hardware system. This value, therefore, is considered to be the lower bound resource requirement for the classes of applications of MI studied in this work.

The main memory capacity requirement parameter is determined, as in the case of FLOPS, by the 3-D facial recognition application. Among the classes of problems considered in this research, the resource requirement in terms of main memory capacity is concluded to be approximately 43GB.

It has just been shown that a crucial parameter for MI is nearly one TFLOPS, which is much more than most modern computing machines are capable of achieving. The result of this analysis suggests that for this particular algorithm (and hence for the field of MI) HPC resources are required. So what are the current and future trends of HPC systems that will be used to support some of the applications of MI? The next section provides a discussion on the current state of the technology and what the future trends on technology advancements may be.

9.2. Future Technology Trends

The two most widely known observations that have been used in the past in order to summarize and predict advancements in the semiconductor industry are Moore's Law and Dennard Scaling.

Moore's Law [**139**] is an observation regarding the number of transistors in an integrated circuit and was later extended to include the speed of those transistors, and therefore the overall performance of microprocessors. At the time of the initial publication, Moore predicted for this law to be in effect for at least another decade. In 1975 he revised the law with a prediction that the number of transistors will double every two years. Moore's law projection stayed in effect until approximately 2012, at which point it began to slow down.

Dennard Scaling, also know as MOSFET scaling [**48**], is a scaling law that states that as transistors get smaller their power density stays constant so that power is in proportion with area. That leads to performance per watt exponentially increasing with smaller transistors thereby leading to faster clock rates. Dennard Scaling stopped being applicable approximately in 2006: in the recent

years, transistors counts continue to grow, but at a slower rate compared to the original observation. The inability to continue the increase of clock frequencies at the rate of years before 2004 caused the CPU manufacturers to seek alternative ways to improve performance, e.g. multiple cores [**103**].

With the change of technology development rates, new efforts have been dedicated to estimating the potential future rate of technology development, including predictions on overall HPC systems performance, memory, and power characteristics and requirements. These efforts provide predictions of a few important factors of technology advancements for MI, including the theoretical system peak performance (R_{peak}), main memory bandwidth, ratio of memory to performance [bytes/FLOP], and energy [Joules/FLOP] [**14**,**105**].

Future trends in the area of hardware systems is summarized below. These results are a combination of predictions of a few studies done in the recent years. Figure 9.2 shows predicted trends based on three models: the 2008 model, as well as Scaled and Constant models. The 2008 model was originally introduced in [**14**], while the later models are revised versions that were based on the original 2008 model. In the Scaled model the inherent energy is improving proportional to the increase in FLOPS, assuming constant chip power, which is an optimistic prediction. In the second model, named Constant model, total energy per access is fixed, which means that power is proportional to the access rate. This is a pessimistic prediction.

Figure 9.2 provides an estimate on the overall system peak performance over several previous years, which is the key metric used to quantify resource requirements for MI.

The red squares represent the peak performance of the top 10 fastest supercomputers according to the Top500 list. Based on this input data, only the optimistic Scaled model predicts the peak performance exaflop system by the year of 2025, while the other two models predict performance ranges on the order of hundreds of petaflops. More importantly, the asymptotic behavior of the projected functions suggest a gradual decrease in the R_{peak}, which is in accordance with the Moore's Law and Dennard scaling and means that the approach using present architectures is nearing its end. Additionally, these results are predicted to be achieved in about 2020 with a requirement that about 1200 racks with the gross power consumption of 180 to 425 MW would be necessary. This is an order of magnitude off as compared to the original plan of an EFLOPS

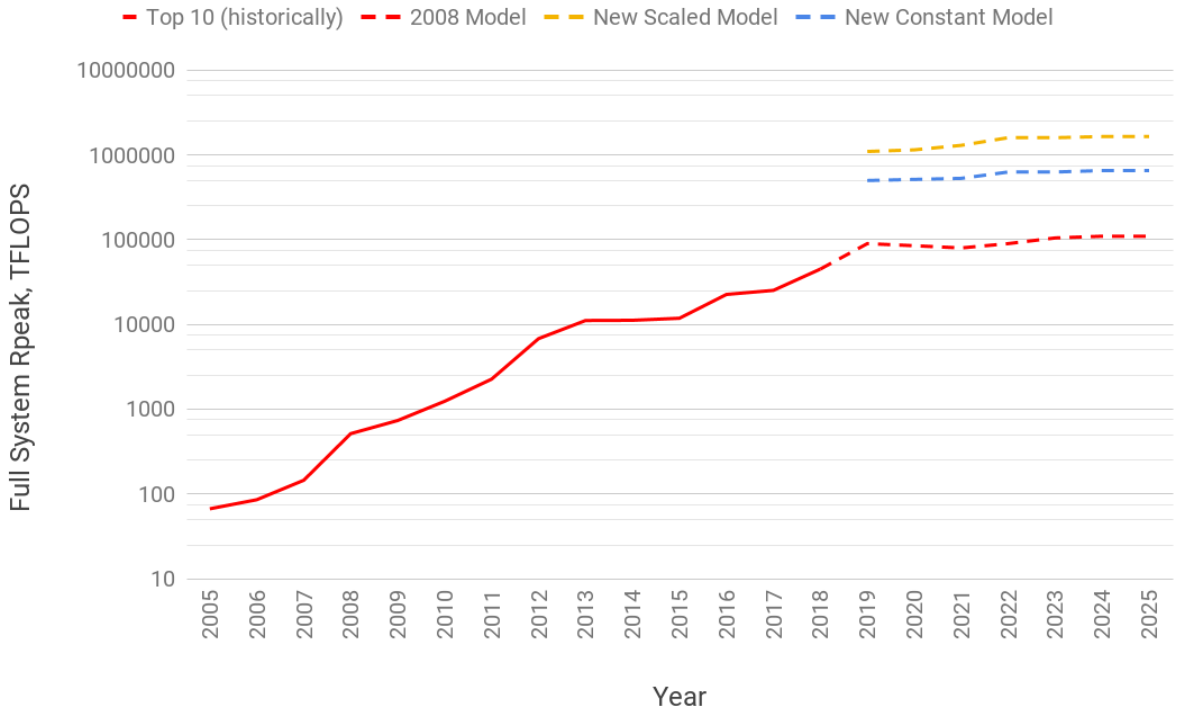

FIGURE 9.2. Full system peak performance (R_{peak}) evolution and future predictions. Red solid line denotes the historic data of the average performance of fastest 10 supercomputers from Top500 list. Three models were used to predict the behavior of R_{peak} in the future. The most conservative model suggests that at the current progress rate the systems on Top500 list will reach hundreds of PFLOPS, while the most optimistic model predicts EFLOPS by approximately 2019. Data taken from [**105**].

system with the power consumption of 20 MW, and is more than twice the power requirements of the fastest supercomputer as of this writing (8.806 MW as of June 2018 Top500 rankings [**185**]).

Another important parameter used in the resource requirements estimation model is main memory. Figure 9.3 shows the historic change and prediction of main memory bandwidth. Again, the peak memory logic bandwidth is predicted to flatten beginning from 2020.

One additional graph describing a trend in current and future HPC systems is the ratio of the amount of memory per FLOPS. As can be seen from Table 9.1, this ratio has been decreasing since 2004. This means that modern machines are built with more floating point operations capacity in them than memory capacity and as the number of FLOPS is growing, memory is getting smaller.

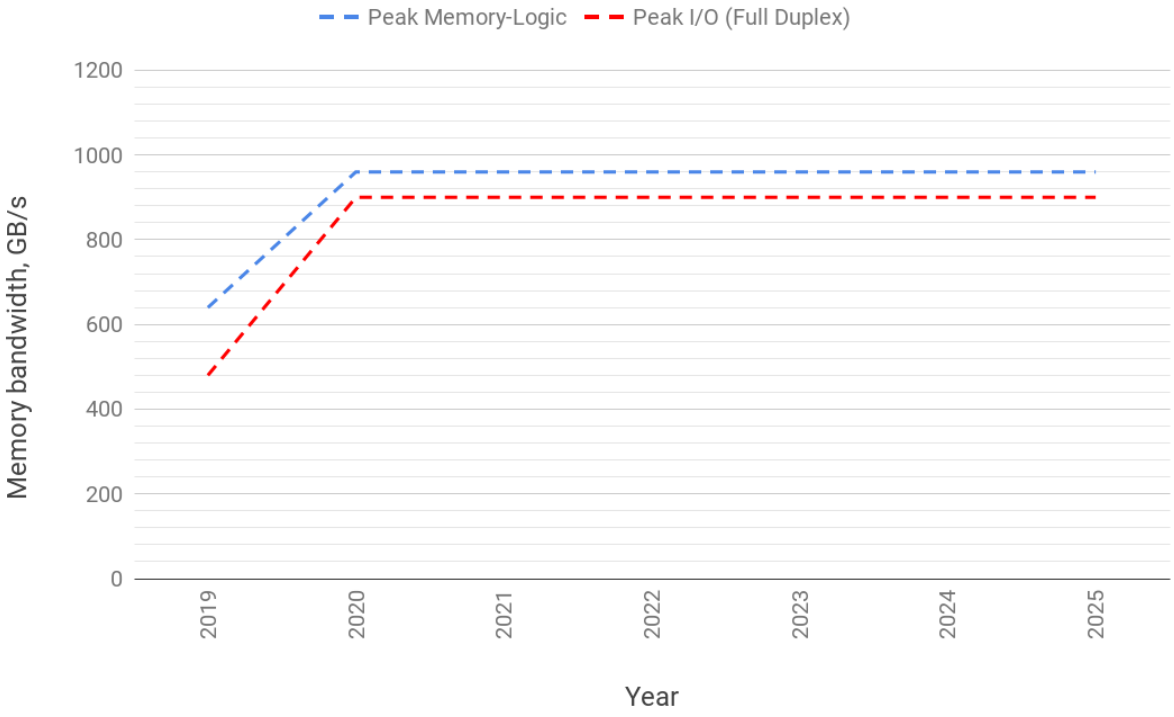

FIGURE 9.3. HPC systems memory bandwidth estimates. Flattening of bandwidth improvement is expected around 2020. Data from **[104]**.

For example, for a modern GPU with 32 GB memory capacity and 125 TFLOPS, this ratio is on the order of $\mathcal{O}(10^{-2})$.

Year	**Ratio of Memory/R_{max} [GB/FLOPS]**
2004	1.5
2008	0.875
2012	0.375
2016	0.075

TABLE 9.1. Memory per FLOPS is dropping. The compound annual growth rate is 0.72. Data taken from **[104]**.

Finally, a trend shows a decrease in energy per Floating Point Operation (FLOP) over time. The projection model predicts the energy trend accurately from 2006 until 2014. If this model

continues to be accurate in the future, it is expected for HPC systems to have between 180 and 420 pJ per FLOP [**104**].

In summary, the predictions for current architecture approaches in building HPC systems suggest that the technology advancements are not going to continue at the rates observed in the past years. All of the parameters and metrics that are considered key for characterizing resource requirements for problems of MI are expected to see a shift in the rate of development of previous years. Specifically, overall maximal performance improvement of supercomputers is expected to observe a limit, as well as the main memory bandwidth and energy per FLOP ration. Most importantly, the amount of memory per FLOP is already seeing a decrease, which means there has been less memory to FLOPS delivered over the past decade.

Now that the current resource requirements along with the future trends in the hardware HPC systems are discussed, the last piece, which is the analysis of both future trends combined is provided. Next Section discusses the trends in the amounts of data that are predicted to be used for solving problems in the field of MI.

9.3. Machine Intelligence Future Trends

This Section provides a prediction of the rate of development for applications and algorithms in the field of Machine Intelligence, as well as the hardware computation systems that will support these algorithms in the future. The predictions about the hardware are drawn from prior research in the field (see Section 9.2), while the predictions about the complexity of applications of MI are based on the outcomes of Section 9.1.

The growth rate of available data for 3-D facial recognition algorithm (as an example of MI) is predicted based on the historic evolution of the amount of available pixels in conventional cameras over years (see Figure 9.4). The blue line shows the historic development of the amount of MPixels over the years, with approximately 3.11 MPixels in 2000 all the way up to 400 MPixels in 2018 (non-conventional Hasselblad H6D-400c with a cost of $48,000 in 2018 [**81**]). Based on the historic data, two models of predicted amount of MPixels vs years are shown, with the pessimistic model projecting approximately 500 MPixels and a little less than 3 GPixels for the optimistic model by the year of 2030.

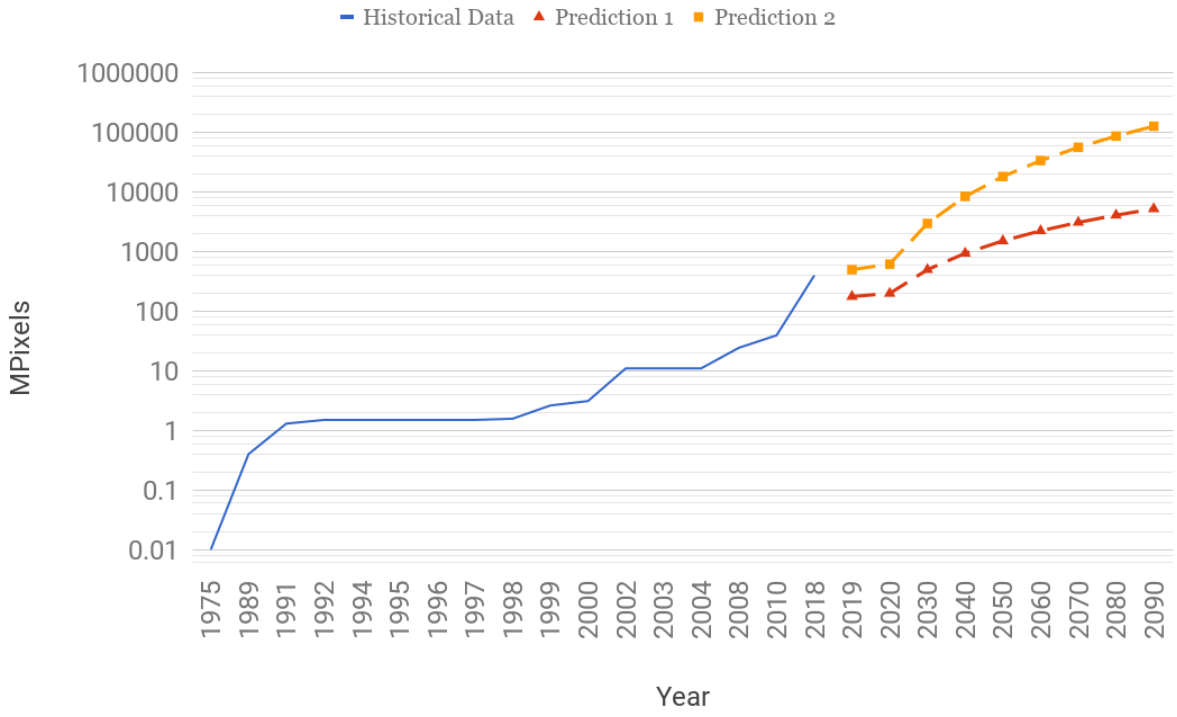

FIGURE 9.4. Mpixels over time + future predictions

Given a known relation between the number of pixels and particular year, the number of Operations (Ops) that would be required for an algorithm to perform 3-D facial recognition analysis on a certain number of pixels can be established (see Figure 9.5). Taking the real-time processing requirement assumption into account (time to solution of approximately one second), the amount of Ops that is required to execute this algorithm can be viewed as a requirement of a number of FLOPS imposed on a machine that is running this problem. The red dotted line indicates the number of Ops required to perform the 3-D facial recognition vs the number of GPixels. The blue star indicates the results of the actual experiment with approximately 5.52×10^{11} Ops achieved for the case of 190 MPixels (1.9×10^{-1} GPixels). All the results of 3-D facial recognition discussed in this Section were obtained using a pointcloud of 4663 points, unless noted otherwise explicitly.

Finally, combining all the results discussed above, Figure 9.6 provides a summary of the predicted growth rates of sensor data for typical applications of MI on an example of 3-D facial recognition, and the growth rates of HPC systems of the future decade. The dark blue and sky blue lines indicate the amount of FLOPS that are required for a system to perform the tasks in real

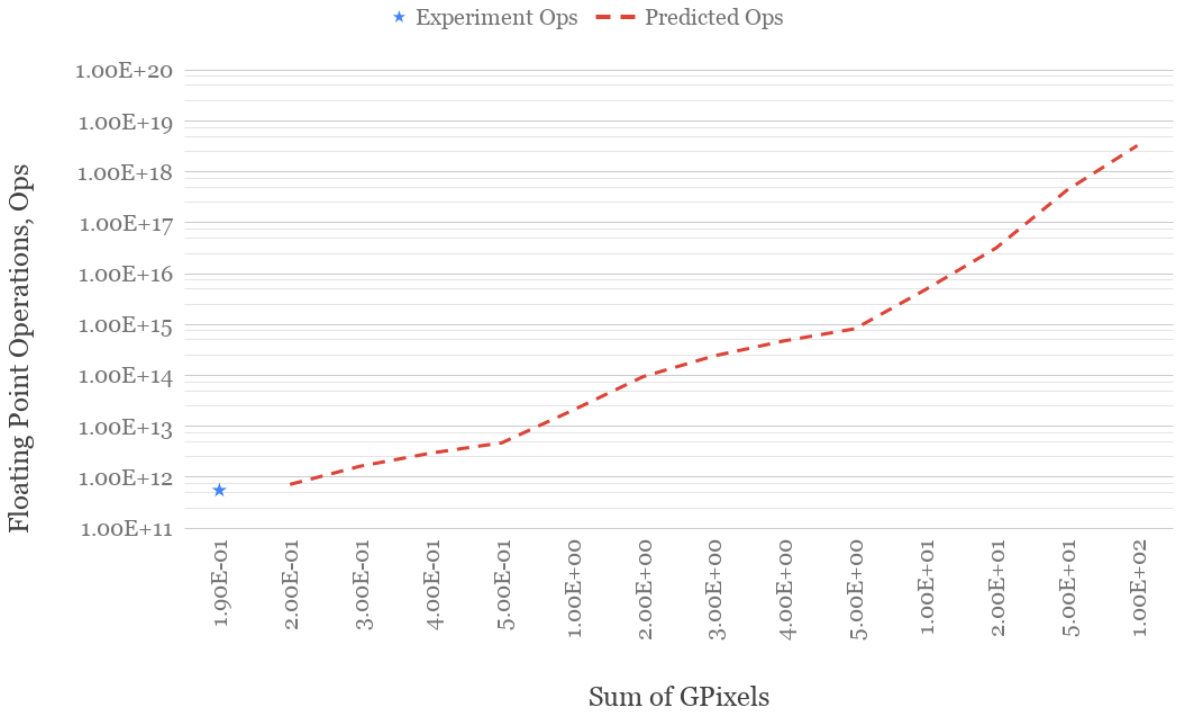

FIGURE 9.5. Ops vs the sum of GPixels for all cameras used in experiment. Red dashed line signifies the projected number of Ops vs the corresponding GPixels. The blue star corresponds to the value of an actual experiment.

time. The sky blue line corresponds to a task of 3-D facial recognition for a lower bound small point cloud (approximately 4.5K points), while the dark blue line provides performance resource requirements for upper bound large point cloud (approximately 4.5M points). Based on these graphs, the resource requirements for a lower bound case are approximately 5.15 TFLOPS, and are projected to grow up to approximately 181 TFLOPS by 2025. The upper bound resource requirements are situated at 552 EFLOPS (552×10^{18}), with the projected requirements of hundreds of ZFLOPS (181×10^{21}) by the year of 2025. These colossal numbers of FLOPS required for the upper bound experiment are obviously not achievable by any of the today's HPC systems. As can be seen from the Figure, the sensor data growth plots are bounded by the blue graphs representing the hardware systems trends. This means that the lower bound experiment is projected to be possible to be performed by any HPC system on Top500 list of the next decade, while it would not be possible to execute the upper bound experiment in real time.

Another important result is that the rates of growth for sensor data for application of MI are increasing at higher rates when compared to the corresponding HPC technology development growth. This means that some of these lines (e.g. the gold (#500 Top500) and sky blue (Ops Small Pointcloud) might intersect beyond the year of 2025, which would mean that the amounts of parallelism of #500 machines on Top500 list would not be sufficient to match the real-time processing requirements of algorithms of MI.

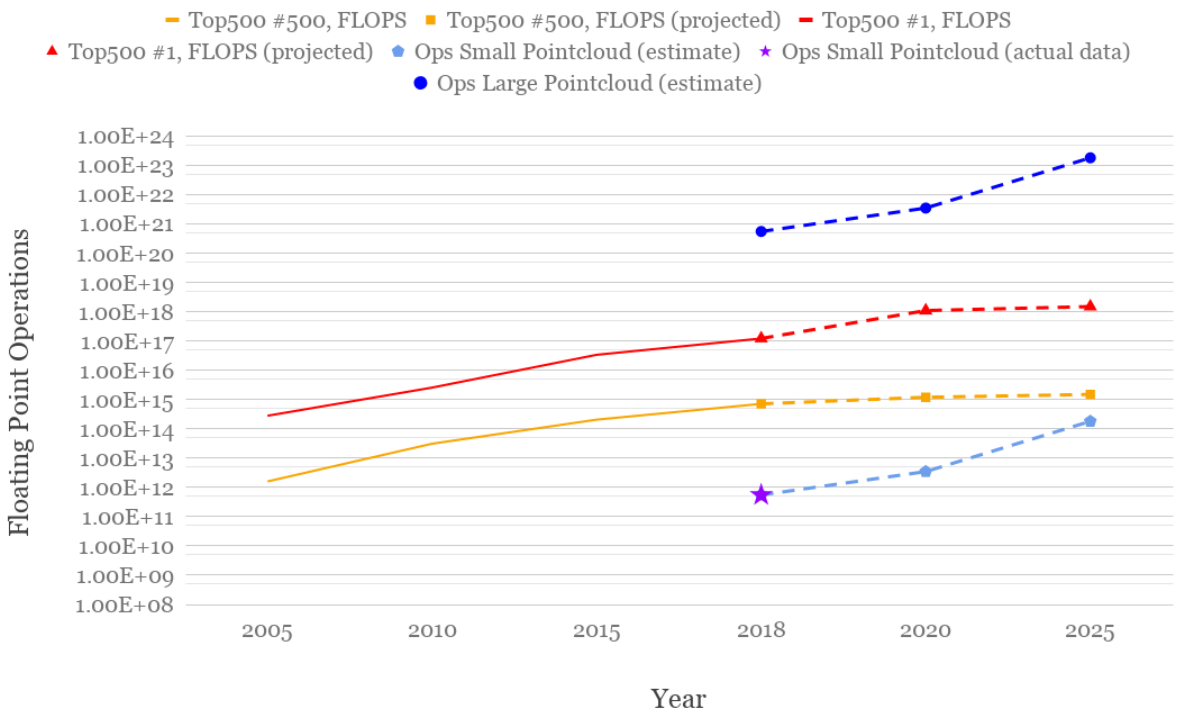

FIGURE 9.6. Predictions for future development of supercomputers peak performance and the amount of sensor data for applications of MI. Red solid line shows the historical performance development of the fastest system on the Top500 list, while the red dashed line provides a projection on the future development of the fastest Top500 system based on **[105]**. Yellow solid and dashed lines correspond to actual and projected FLOPS respectively. Sky blue line provides a projection of the number of FLOPS required for a lower bound problem real-time processing. The purple star signifies the amount of FLOPS obtained experimentally. The dark blue line signifies the amount of FLOPS required for a hypothetical HPC system that is to process the upper bound MI problem in real time.

9.4. Summary

In summary, current resource requirements for the systems of Machine Intelligence operating in real-time are approximately 0.63 TFLOPS, and 0.55 TFLOPS for time to solutions of 0.89 and 1 second respectively. The technology development that is affecting the amount of parallelism of modern and future HPC systems is experiencing a reduced rate of growth and is expected to flatline in the next decade. The amounts of sensor data that are processed by typical Machine Intelligent agents are experiencing growth and are projected to grow at much faster rates, as compared to the amount of exploitable parallelism of hardware systems. This scenario suggests that at the rate these developments are proceeding, the future HPC systems may not be sufficient to process the amounts of data of applications for Machine Intelligence in real time.

10
Conclusions

This thesis introduced an approach to defining a lower bound on resource requirements for Machine Intelligence (MI). It introduced a definition of MI, as well as a model, named the Cognitive Real-time Interactive System (CRIS), that provides a detailed description of an architecture for a system of MI, as well as how such a system operates by defining a workflow, and finally what the associated costs of its components are by defining a performance model. Furthermore, external driver applications that define CRIS's behavior were introduced with the purpose of estimating the resource requirements of particular workflow elements. One of the considered external drivers was novel in the field of MI by providing an approach to solving a problem of 3-D facial recognition with the requirement of real-time processing utilizing High Performance Computing (HPC) resources. Additionally, evaluation of the amounts of data and HPC resources that are required today to support the real-time processing in the field of MI were provided. Projections of how the availability of HPC resources and the demands of near-term future MI algorithms relate have been performed. The result of this analysis suggests that the MI requirements of the near future are set to exceed the available HPC resources.

10.1. Summary of results

This thesis aims to answer the following question:

"What is the lower bound on resource requirements for Machine Intelligence?"

In order to provide an answer to this question, a few objectives had to be satisfied.

First, a working definition of what constitutes Machine Intelligence (within the boundaries of this research study) was introduced.

Second, according to that definition, an abstract architecture that describes classes of problems of Machine Intelligence, as well as a generic Workflow that provides details of how the elements of the architecture interact with each other were introduced.

Third, a performance model, including a metric and a requirement parameter that both quantify the resource requirements for MI were introduced. Evidence was provided that the present performance model is able to characterize known applications of MI in terms of Floating Point Operations per Second (FLOPS) and the amount of main memory required. Further, it was shown on particular examples how the performance model can be applied to complex problems that incorporate a variety of simpler tasks, or to problems that are combinations of various independents tasks of MI.

Fourth, three external driver applications that are placed on top of the generic workflow were described. Experiments with those driver applications were performed that allowed to generate the achieved performance [FLOPS], and main memory [GB] requirements in order for current computer systems to support these particular applications of MI. The results obtained established the lower bound resource requirements for today's applications of MI: the requirement for the main memory parameter was established at approximately 43 GB, while the requirement on achieved performance that the hardware system should yield was established to be approximately 0.63 TFLOPS (6.3×10^{11} FLOPS).

10.2. Future work

As the field of MI continues to grow, along with the advancements in technology, HPC, and special-purpose Application-Specific Integrated Circuits (ASICs), the possibilities for future work in this field are plenty. The most important ones are described below.

First, the abstract architecture for MI can be expanded to include a specification supporting learning. Learning and machine understanding, as two crucial concepts in the field, can be defined as finite algorithms that are part of the specification of the abstract architecture. This definition will include interrelation of objective function and active context stack, combined with current and future approaches in Artificial Intelligence (AI) and Machine Learning (ML).

Second, the performance model can be expanded by incorporating various additional metrics and requirement parameters to quantify the problems of MI in a more thorough manner. Some

examples of additional metrics are Traversed Edges per Second (TEPS), network latency and bandwidth, I/O rates, etc.

Finally, as the algorithms in the field of MI become more complex with increased amounts of data processed by the systems, new driver applications can be used in order to stimulate the CRIS workflow. The modern cutting edge MI applications already include systems of significant complexities, e.g. personal assistants on mobile devices or computers that require solutions to a combination of problems, e.g. natural languages + facial recognition. The complexities of problems of MI will only increase, thereby requiring careful considerations with regards to the amounts of computational resources available.

10.3. Final thoughts

The field of Machine Intelligence, although not new, has been attracting significant attention in the past decade and is expected to continue growing at even faster rates. As the results of this study show, the trends of the recent technology development and the advancements ing the field of MI suggest that rates of growth for the two fields do not agree. This problem can be addressed by creating special purpose hardware, such as ASICs, that would be able to provide adequate performance capabilities to the increasing demands of new real-time MI systems. The results and techniques described in this work can be used in order to predict future trends of these two areas, and to guide the directions in which the future advancements can be accomplished.

Bibliography

[1] AI Index Report. http://cdn.aiindex.org/2017-report.pdf, 2017.

[2] The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems. AIPS-00 Planning Competition. http://ipc00.icaps-conference.org/.

[3] James S Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):473–509, 1991.

[4] Allwords Dictionary. https://www.allwords.com.

[5] M. Anderson, T. Sterling, and B. Zhang. Method and apparatus for 3-d facial recognition, 2018. Pub. No. US 2018/0165510 A1.

[6] M. A. Anusuya and S. K. Katti. Speech Recognition by Machine, A Review. *CoRR*, abs/1001.2267, 2010.

[7] Itamar Arel, Derek C. Rose, and Thomas P. Karnowski. Research frontier: Deep machine learning–a new frontier in artificial intelligence research. *Comp. Intell. Mag.*, 5(4):13–18, November 2010.

[8] John V Arthur, Paul A Merolla, Filipp Akopyan, Rodrigo Alvarez, Andrew Cassidy, Shyamal Chandra, Steven K Esser, Nabil Imam, William Risk, Daniel BD Rubin, et al. Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[9] Yannis M Assael, Brendan Shillingford, Shimon Whiteson, and Nando de Freitas. Lipnet: Sentence-level lipreading. *arXiv preprint*, 2016.

[10] Mehdi Assefi, Guangchi Liu, Mike P Wittie, and Clemente Izurieta. An experimental evaluation of apple siri and google speech recognition. *Proccedings of the 2015 ISCA SEDE*, 2015.

[11] Fahiem Bacchus. Aips 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems. *Ai magazine*, 22(3):47, 2001.

[12] Jerome R Bellegarda. Large vocabulary speech recognition with multispan statistical language models. *IEEE Transactions on Speech and Audio Processing*, 8(1):76–84, 2000.

[13] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards AI. *Large-scale kernel machines*, 34(5), 2007.

[14] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving

exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15, 2008.

[15] Jean-Philippe Bernardy and Koen Claessen. Efficient divide-and-conquer parsing of practical context-free languages. In *ACM SIGPLAN Notices*, volume 48, pages 111–122. ACM, 2013.

[16] Walter Van Dyke Bingham et al. Aptitudes and aptitude testing. In *National Occupational Conference (US)*. Pub. for the National occupational conference by Harper & brothers, 1942.

[17] Christopher M Bishop. Pattern recognition. *Machine Learning*, 2006.

[18] L Susan Blackford, Jaeyoung Choi, Andy Cleary, Eduardo D'Azevedo, James Demmel, Inderjit Dhillon, Jack Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, et al. *ScaLAPACK users' guide*. SIAM, 1997.

[19] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *IJCAI*, pages 1636–1642, 1995.

[20] Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300, 1997.

[21] Bernd Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 89–97. Association for Computational Linguistics, 2010.

[22] Alan H Bond and Les Gasser. *Readings in distributed artificial intelligence*. Morgan Kaufmann, 2014.

[23] A. Borgida and John F. Sowa. *Principles of semantic networks: explorations in the representation of knowledge*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1991.

[24] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[25] Encyclopedia Britannica. `https://www.britannica.com`.

[26] AE Bryson and YC Ho. Applied optimal control (blaisdell, waltham, ma). *Google Scholar*, 1969.

[27] Christian Buck, Kenneth Heafield, and Bas Van Ooyen. N-gram counts and language models from the common crawl. In *LREC*, volume 2, page 4. Citeseer, 2014.

[28] Burroughs B5000 Information Brochure. `http://www.cs.virginia.edu/brochure/images/manuals/b5000/brochure/b5000_broch.html`.

[29] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[30] Emmanuel Candes, Laurent Demanet, David Donoho, and Lexing Ying. Fast discrete curvelet transforms. *Multiscale Modeling & Simulation*, 5(3):861–899, 2006.

[31] Jaime Guillermo Carbonell. Subjective understanding: Computer models of belief systems. Technical report, Yale University New Haven Conn Department of Computer Science, 1979.

[32] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.

[33] Xavier Carreras. Experiments with a higher-order projective dependency parser. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[34] Bruce Char, Keith Geddes, and Gaston Gonnet. The maple symbolic computation system. *SIGSAM Bull.*, 17(3-4):31–42, August 1983.

[35] Bruce Char, Keith O Geddes, Gaston H Gonnet, Benton L Leong, Michael B Monagan, and Stephen Watt. *Maple V library reference manual*. Springer Science & Business Media, 2013.

[36] Huaijin G Chen, Suren Jayasuriya, Jiyue Yang, Judy Stephen, Sriram Sivaramakrishnan, Ashok Veeraraghavan, and Alyosha Molnar. Asp vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 903–912, 2016.

[37] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[38] Jaeyoung Choi, Jack J Dongarra, Roldan Pozo, and David W Walker. Scalapack: A scalable linear algebra library for distributed memory concurrent computers. In *Frontiers of Massively Parallel Computation, 1992., Fourth Symposium on the*, pages 120–127. IEEE, 1992.

[39] William Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003.

[40] Collins dictionary. `https://www.collinsdictionary.com`.

[41] Martin Cooke, Phil Green, Ljubomir Josifovski, and Ascension Vizinho. Robust automatic speech recognition with missing and unreliable acoustic data. *Speech communication*, 34(3):267–285, 2001.

[42] Eric Crawford, Matthew Gingerich, and Chris Eliasmith. Biologically plausible, human-scale knowledge representation. *Cognitive science*, 40(4):782–821, 2016.

[43] Richard Edward Cullingford. Script application: computer understanding of newspaper stories. Technical report, Yale University New Haven Conn Department of Computer Science, 1978.

[44] William Cushing, J Benton, and Subbarao Kambhampati. Replanning as a deliberative re-selection of objectives. *Arizona State University CSE Department TR*, 2008.

[45] Prudhvi Raj Dachapally and Michael N Jones. Catastrophic interference in neural embedding models. In *Proceedings of the 40th Annual Meeting of the Cognitive Science Society*, 2018.

[46] Robert Michael S. Dean. Common world model for unmanned systems. *Proc. SPIE*, 8741:87410O–87410O–9, 2013.

[47] José G Delgado-Frias and Will Moore. *VLSI for artificial intelligence*, volume 68. Springer Science & Business Media, 2012.

[48] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[49] Patrick Doherty, Witold Lukaszewicz, Andrzej Skowron, and Andrzej Szalas. *Knowledge Representation Techniques - A Rough Set Approach*, volume 202 of *Studies in Fuzziness and Soft Computing*. Springer, 2006.

[50] Nira Dyn and David Levin. Iterative solution of systems originating from integral equations and surface interpolation. *SIAM Journal on Numerical Analysis*, 20(2):377–390, 1983.

[51] Stefan Edelkamp and Jörg Hoffmann. Pddl2. 2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC04), at ICAPS04*, 2004.

[52] Facial Recognition Market — Worldwide Manufacturing Growth, Trends Forecast 2022, Dec 17 2016. Copyright - 2016 Global Data Point. All Rights Reserved. Provided by SyndiGate Media Inc. (Syndigate.info); Last updated - 2016-12-17.

[53] Scott Fahlman. Computing facilities for ai: a survey of present and near-future options. *AI Magazine*, 2(1):16, 1981.

[54] Scott E Fahlman. An empirical study of learning speed in back-propagation networks. *Technical Report*, 1988.

[55] James Fan, Aditya Kalyanpur, DC Gondek, and David A Ferrucci. Automatic knowledge extraction from documents. *IBM Journal of Research and Development*, 56(3.4):5–1, 2012.

[56] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 109–116. IEEE, 2011.

[57] Edward A Feigenbaum and Pamela McCorduck. *The fifth generation*. Addison-Wesley Pub., 1983.

[58] D. A. Ferrucci. Introduction to this is watson. *IBM Journal of Research and Development*, 56(3.4):1:1–1:15, May 2012.

[59] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[60] Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[61] Razvan V Florian. Autonomous artificial intelligent agents. *Technical Report Coneural-03–01*, 2003.

[62] Michael J. Frisch. Remarks on algorithm 352 [s22], algorithm 385 [s13], algorithm 392 [d3]. *Commun. ACM*, 15(12):1074–, December 1972.

[63] Paul Furgale, Ulrich Schwesinger, Martin Rufli, Wojciech Derendarz, Hugo Grimmett, Peter Mühlfellner, Stefan Wonneberger, Julian Timpner, Stephan Rottmann, Bo Li, et al. Toward automated driving in cities using close-to-market sensors: An overview of the v-charge project. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 809–816. IEEE, 2013.

[64] Alan Garvey and Victor Lesser. A survey of research in deliberative real-time artificial intelligence. *The Journal of Real-Time Systems*, 6:317–347, 1993.

[65] Timur Gilmanov and Maciej Brodowicz. An interactive natural language understanding and planning system: COMMAND.PLAN.EXECUTE, November 2013. Presentation in the Indiana Unviersity booth at SC 2013.

[66] Alessandro Giusti, Dan C Ciresan, Jonathan Masci, Luca M Gambardella, and Jurgen Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 4034–4038. IEEE, 2013.

[67] A Gliozzo, O Biran, S Patwardhan, and K McKeown. Semantic technologies in IBM watson. In *Proceedings of the fourth workshop on teaching NLP and CL*, pages 85–92, 2013.

[68] Ben Goertzel. The hidden pattern. *Brown Walker*, 2006.

[69] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[70] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[71] Google Inc. `cloud.google.com/speech/`.

[72] Linda S Gottfredson. Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography, 1997.

[73] Graph 500 List. `http://graph500.org/?page_id=382`.

[74] Green 500 List. `https://www.top500.org/green500/lists/2018/06`.

[75] Samuel Greengard. Making chips smarter. *Commun. ACM*, 60(5):13–15, April 2017.

[76] Richard L Gregory and Oliver Louis Zangwill. *The Oxford companion to the mind*. Oxford University Press, 1987.

[77] Patrick Grother, Mei Ngan, and Kayee Hanaoka. Ongoing Face Recognition Vendor Test (FRVT). Technical report, National Institute of Standards and Technology (NIST), 2018.

[78] R. Guha, Rob McCool, and Eric Miller. Semantic search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 700–709, New York, NY, USA, 2003. ACM.

[79] Erico Guizzo. How google?s self-driving car works. *IEEE Spectrum Online, October*, 18, 2011.

[80] William H Harris, Judith S Levey, et al. *New Columbia encyclopedia*. Columbia University Press: distributed by Lippincott, 1975.

[81] Hasselblad Press Release. Hasselblad introduces the h6d-400c. `https://www.hasselblad.com/press/press-releases/hasselblad-introduces-the-h6d-400c-ms`, 2018.

[82] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[83] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009.

[84] James Hays and Alexei A Efros. Scene completion using millions of photographs. In *ACM Transactions on Graphics (TOG)*, volume 26, page 4. ACM, 2007.

[85] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[86] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26:191–246, 2006.

[87] Malte Helmert. On the complexity of planning in transportation domains. In *Sixth European Conference on Planning*, 2014.

[88] James A Hendler, Austin Tate, and Mark Drummond. AI planning: Systems and techniques. *AI magazine*, 11(2):61, 1990.

[89] W Daniel Hillis. *The connection machine*. MIT press, 1989.

[90] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[91] Albert Sydney Hornby, Edward Vivian Gatenby, and Harold Wakefield. *The advanced learner's dictionary of current English*, volume 965. Oxford University Press London, 1963.

[92] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[93] Random House. *Random House Webster's unabridged dictionary*. Random House Reference, 2001.

[94] Miloš Jakubíček, Adam Kilgarriff, Diana McCarthy, and Pavel Rychlỳ. Fast syntactic searching in very large corpora for many languages. In *Proceedings of the 24th Pacific Asia Conference on Language, Information and Computation*, 2010.

[95] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 1–12. IEEE, 2017.

[96] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Draft, 2017.

[97] Steve Kaufmann and Bill Homer. Craypat-cray x1 performance analysis tool. *Cray User Group (May 2003)*, 2003.

[98] Henry Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999.

[99] Adam Kilgarriff and Gregory Grefenstette. Introduction to the special issue on the web as corpus. *Computational linguistics*, 29(3):333–347, 2003.

[100] Akihiro Kishimoto, Alex S Fukunaga, Adi Botea, et al. Scalable, parallel best-first search for optimal sequential planning. In *ICAPS*, pages 201–208, 2009.

[101] Hiroaki Kitano and James A Hendler. *Massively parallel artificial intelligence*. Aaai Press, 1994.

[102] Johan de Kleer, Jon Doyle, Guy L. Steele, Jr., and Gerald Jay Sussman. Amord explicit control of reasoning. *SIGPLAN Not.*, 12(8):116–125, August 1977.

[103] Peter Kogge and John Shalf. Exascale computing trends: Adjusting to the" new normal" for computer architecture. *Computing in Science & Engineering*, 15(6):16–26, 2013.

[104] Peter M Kogge. Reading the tea-leaves: How architecture has evolved at the high end. In *IPDPS*, page 515, 2014.

[105] Peter M Kogge. Updating the energy model for future exascale systems. In *International Conference on High Performance Computing*, pages 323–339. Springer, 2015.

[106] Teuvo Kohonen. *Content-addressable memories*, volume 1. Springer Science & Business Media, 2012.

[107] Teuvo Kohonen and Panu Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, 21(1):19–30, 1998.

[108] Janusz S Kowalik. *Parallel computation and computers for artificial intelligence*, volume 26. Springer Science & Business Media, 2012.

[109] Ray Kurzweil. *The age of spiritual machines: When computers exceed human intelligence*. Penguin, 2000.

[110] John E Laird. Extending the soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, 171:224, 2008.

[111] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[112] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[113] Kai-Fu Lee. *Automatic speech recognition: the development of the SPHINX system*, volume 62. Springer Science & Business Media, 1988.

[114] Shane Legg and Marcus Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, 2007.

[115] Shane Legg, Marcus Hutter, et al. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

[116] Yuangui Lei, Victoria Uren, and Enrico Motta. Semsearch: A search engine for the semantic web. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 238–245. Springer, 2006.

[117] D Lenat and E Feigenbaum. On the thresholds of knowledge. *Foundations of Artificial Intelligence, MIT Press, Cambridge, MA*, pages 185–250, 1992.

[118] Douglas Lenat and Ramanathan V Guha. Cyc: A midterm report. *AI magazine*, 11(3):32, 1990.

[119] Chu Min Li. Heuristics based on unit propagation for satisfiability problems. In *In Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 366–371, 1997.

[120] J. Lighthill, N. S. Sutherland, R. M. Need-ham, H. C. Longuet-Higgins, and D. Michie. Artificial intelligence: A general survey. In *Artificial Intelligence: A Paper Symposium. London: Science Research Council*, 1973.

[121] Longman dictionary of contemporary english, 2018.

[122] Moshe Looks, Ben Goertzel, and Cassio Pennachin. Novamente: An integrative architecture for general intelligence. In *AAAI fall symposium, achieving human-level intelligence*, 2004.

[123] Madagascar Planner Webpage. `https://research.ics.aalto.fi/software/sat/madagascar`.

[124] Tanaya Mandal, QM Jonathan Wu, and Yuan Yuan. Curvelet based face recognition via dimension reduction. *Signal Processing*, 89(12):2345–2353, 2009.

[125] William A Martin and Richard J Fateman. The macsyma system. In *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 59–75. ACM, 1971.

[126] J. McCarthy. What is artificial intelligence? `http://www-formal.stanford.edu/jmc/whatisai/whatisai.html`, 2004.

[127] John McCarthy. *Programs with common sense*. RLE and MIT Computation Center, 1960.

[128] Michael C McCord, J William Murdock, and Branimir K Boguraev. Deep parsing in watson. *IBM Journal of Research and Development*, 56(3.4):3–1, 2012.

[129] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1):21–39, Aug 2012.

[130] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1–4. IEEE, 2011.

[131] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[132] Inc Merriam-Webster. *Webster's ninth new collegiate dictionary*. Merriam-Webster, 1983.

[133] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[134] Marvin L Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

[135] Mobile Linpack Results. `http://linpack.hpc.msu.ru/index.php`.

[136] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[137] David A Moon. Architecture of the symbolics 3600. *ACM SIGARCH Computer Architecture News*, 13(3):76–83, 1985.

[138] Bert Moons, Marian Verhelst, et al. Resource aware design of a deep convolutional-recurrent neural network for speech recognition through audio-visual sensor fusion. *arXiv preprint arXiv:1803.04840*, 2018.

[139] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[140] William Morris et al. *American heritage dictionary of the English language*. American heritage, 1969.

[141] Joel Moses. Macsyma: A personal history. *Journal of Symbolic Computation*, 47(2):123 – 130, 2012.

[142] Tohru Moto-Oka. *Fifth generation computer systems*. Elsevier, 2012.

[143] Erik T Mueller. *Commonsense Reasoning: An Event Calculus Based Approach*. Morgan Kaufmann, 2014.

[144] Hideyuki Nakashima. Ai as complex information processing. *Minds and machines*, 9(1):57–80, 1999.

[145] Ulric Neisser, Gwyneth Boodoo, Thomas J Bouchard Jr, A Wade Boykin, Nathan Brody, Stephen J Ceci, Diane F Halpern, John C Loehlin, Robert Perloff, Robert J Sternberg, et al. Intelligence: Knowns and unknowns. *American psychologist*, 51(2):77, 1996.

[146] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[147] Sergei Nirenburg and Victor Raskin. *Ontological semantics*. MIT Press, 2004.

[148] NVIDIA's NVDLA Deep Learning Accelerator Architecture. `http://nvdla.org/contents.html`.

[149] NVIDIA. Nvidia jetson tk1 technical specifications. https://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html, 2018.

[150] Obermayer, H Heller, H Ritter, and K Schulten. Simulation of self-organizing neural nets: A comparison between a transputer ring and a connection machine cm-2. In *Transputer Research and Applications 3: NATUG-3: Proceedings of the Third Conference of the North American Transputer Users Group, April 26-27, 1990, Sunnyvale, CA*, volume 3, page 95. IOS Press, 1990.

[151] Jean Oh, Arne Supp, Felix Duvallet, Abdeslam Boularias, Luis Navarro-Serment, Martial Hebert, Anthony Stentz, Jerry Vinokurov, Oscar Romero, Christian Lebiere, and Robert Dean. Toward mobile robots reasoning like humans. In *AAAI Conference on Artificial Intelligence*, 2015.

[152] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[153] Michael Paluszek and Stephanie Thomas. Generalized 3D spacecraft proximity path planning using A*. In *Infotech@ Aerospace*, page 7043. American Institute of Aeronautics and Astronautics, 2005.

[154] R Peter Bonasso, R James Firby, Erann Gat, David Kortenkamp, David P Miller, and Mark G Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):237–256, 1997.

[155] Antoine Petitet. Hpl-a portable implementation of the high-performance linpack benchmark for distributed-memory computers. *http://www. netlib. org/benchmark/hpl/*, 2004.

[156] Antoine Petitet, Susan Blackford, Jack Dongarra, Brett Ellis, Graham Fagg, Kenneth Roche, and Sathish Vadhiyar. Numerical libraries and the grid. *The International Journal of High Performance Computing Applications*, 15(4):359–374, 2001.

[157] David Lynton Poole, Alan K Mackworth, and Randy Goebel. *Computational intelligence: a logical approach*, volume 1. Oxford University Press New York, 1998.

[158] Dimitrios Prountzos, Roman Manevich, and Keshav Pingali. Synthesizing parallel graph programs via automated planning. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 533–544. ACM, 2015.

[159] Lawrence Rabiner and B Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[160] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[161] Silvia Richter and Matthias Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

[162] Jussi Rintanen. Engineering efficient planners with sat. In *ECAI*, volume 242, pages 684–689, 2012.

[163] Jussi Rintanen et al. Heuristics for planning with sat and expressive action definitions. In *ICAPS*, 2011.

[164] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 10 1986.

[165] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 3 edition, December 2009.

[166] Roger C Schank and Robert P Abelson. Scripts, plans, and knowledge. In *IJCAI*, pages 151–157, 1975.

[167] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[168] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[169] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *In Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-94*, pages 337–343, 1994.

[170] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[171] Herbert Simon. The shape of automation for men and management. *Harper and Row*, 40, 1965.

[172] Apple Inc. Siri. `www.apple.com/ios/siri/`.

[173] Catherine Soanes and Sara Hawker. Compact oxford english dictionary, 2014.

[174] Anne H Soukhanov and Anne H Soukhanov. *Encarta world English dictionary*. St. Martin's Press New York, 1999.

[175] Luc Steels. A self-organizing spatial vocabulary. *Artificial life*, 2(3):319–332, 1995.

[176] T. Sterling, M. Anderson, and M. Brodowicz. *High Performance Computing: Modern Systems and Practices*. Elsevier Science, 2017.

[177] Thomas Sterling, Maciej Brodowicz, and Timur Gilmanov. Towards brain-inspired system architectures. In *Brain-Inspired Computing*, volume 8603 of *Lecture Notes in Computer Science*, pages 159–170. Springer International Publishing, 2014.

[178] Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and scalability of gpu-based convolutional neural networks. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 317–324. IEEE, 2010.

[179] Erich Strohmaier. Top500 supercomputer. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[180] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*, 2017.

[181] Niket Tandon, Bhavana Dalvi Mishra, Joel Grus, Wen-tau Yih, Antoine Bosselut, and Peter Clark. Reasoning about actions and state changes by injecting commonsense knowledge. *arXiv preprint arXiv:1808.10012*, 2018.

[182] Niket Tandon, Aparna S Varde, and Gerard de Melo. Commonsense knowledge in machine intelligence. *ACM SIGMOD Record*, 46(4):49–52, 2018.

[183] Girma S Tewolde. Sensor and network technology for intelligent transportation systems. In *Electro/Information Technology (EIT), 2012 IEEE International Conference on*, pages 1–7. IEEE, 2012.

[184] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[185] Top 500 List. https://www.top500.org/lists/2018/06.

[186] Lewis W Tucker and George G Robertson. Architecture and applications of the connection machine. *Computer*, 21(8):26–38, 1988.

[187] C. Urmson and W. Whittaker. Self-driving cars and the urban challenge. *IEEE Intelligent Systems*, 23(2):66–68, March 2008.

[188] ST VL53L0X Product Specifications. https://www.st.com/resource/en/datasheet/vl53l0x.pdf.

[189] David L Waltz and Jordan B Pollack. Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9(1):51–74, 1985.

[190] Huazheng Wang, Fei Tian, Bin Gao, Jiang Bian, and Tie-Yan Liu. Solving verbal comprehension questions in iq test by knowledge-powered word embedding. *arXiv preprint arXiv:1505.07909*, 2015.

[191] Nigel Ward and David DeVault. Challenges in building highly interactive dialogue systems. *AI Magazine*, 37(4):7–18, 2017.

[192] Joseph Weizenbaum. ELIZA — a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966.

[193] Wikipedia. https://en.wikipedia.org.

[194] Wiktionary. https://www.wiktionary.org, 2006.

[195] David E Wilkins et al. A call for knowledge-based planning. *AI magazine*, 22(1):99, 2001.

[196] Terry Winograd. *Procedures as a representation for data in a computer program for understanding natural language*. PhD thesis, Massachusetts Institute of Technology, 1970.

[197] Terry Winograd. *Understanding Natural Language*. Academic Press, 1972.

[198] Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1):1–191, January 1972.

[199] Wordsmyth dictionary. https://www.wordsmyth.net.

[200] World book encyclopedia dictionary. https://www.worldbook.com.

[201] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.

[202] Kai Zhong, Prateek Jain, and Ashish Kapoor. Fast second-order cone programming for safe mission planning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 79–86. IEEE, 2017.

Timur A. Gilmanov

CONTACT INFORMATION

Indiana University
School of Informatics,
Computing, and Engineering
Smith Research Center 163-20
2805 East 10th Street
Bloomington, IN 47408 USA

E-mail: timugilm@iu.edu
Mobile: +1-225-938-6600

EDUCATION

Indiana University, Bloomington, IN USA

Ph.D., Computer Science, August 2011–November 2018

- 90+/90 credit hours of coursework for Ph.D. degree completed, GPA: 3.93
- Performed research on resource requirements for systems of machine intelligence
- Advisor: Professor Thomas L. Sterling

Louisiana State University, Baton Rouge, LA USA

Ph.D., Computer Science, August 2009–August 2011 (transferred to a Ph.D. program in Computer Science at Indiana University)

- 27/36 credit hours of coursework for Ph.D. degree completed, GPA: 3.72
- Performed research on knowledge management and understanding systems
- Advisor: Professor Thomas L. Sterling

Louisiana State University, Baton Rouge, LA USA

Ph.D., Mechanical Engineering, January 2008–July 2009 (transferred to a Ph.D. program in Computer Science, LSU)

- 15/36 credit hours of coursework for Ph.D. degree completed, GPA: 3.66
- Performed research on computational fluid dynamics and solid body simulations
- Advisor: Professor Sumanta Acharya

Kazan State University, Kazan, Tatarstan, Russia

B.S. and M.S., Applied Mathematics and Information Science, June 2007

- With Honors, GPA: 4.72/5
- Computational mathematics (emphasis on numerical methods and approaches for solving computational fluid dynamics problems)
- Thesis: "An Implicit Algorithm for Solving 3D Equations of Solid Deformable Body Using Material Points Method"

EXPERIENCE

Research Assistant

Research Assistant at the Center for Research in Extreme Scale Technologies and School of Informatics, Computing, and Engineering August 2011–December 2018; Research Assistant at the Center for Computation and Technology, August 2009–July 2011

- Conducted research for measuring resource requirements for the systems of artificial/machine intelligence

Research Assistant at the Department of Mechanical Engineering, January 2008–August 2009

- Conducted research, development, verification, and validation of parallel numerical applications for solving Computational Fluid Dynamics problems using programming language FORTRAN, and libraries PETSc, and MPI

Projects

- Worked on software development of a project named "SWIFT aligner" in the area of Computational Linguistics. Implemented the project from scratch in Java. Introduced features not supported by existing software in this field (visual parallel corpora alignment, various parallel corpora formats import/export, see publications for more details).
- Worked in a team on a project named "ParalleX File System". Implemented a parallel version of a existing serial version of a "merge graph" algorithm. Obtained results of parallel file I/O that are targeted to measure the performance of HPX-3 C++ library combined with POSIX and Orange FS I/O libraries.
- Worked on development and software implementation of various projects in areas including High-performance Computing (Fortran, C++, Python), Software Engineering and Programming Language Concepts (Java), Compilers design and implementation (Java), Natural Language Processing and Computational Linguistics (Python, Java, C++).

Technical Skills

Computer Programming

- Programming languages (sorted by familiarity, most familiar first):
  Python, C++, Java, MATLAB, Scheme, Fortran
- Libraries: MPI, OpenMP, HPX-3, HPX-5, Boost, NLTK

Productivity Applications

- TEX, vi, most common productivity packages for major Operating Systems

Operating Systems

- Mac OS X, Linux, Microsoft Windows family

Teaching Experience

Louisiana State University, Baton Rouge, LA USA
Indiana University, Bloomington, IN USA

Teaching Assistant Multiple times (10+ courses), January 2010 — December 2018

- Introduction to High Performance Computing. Taught students course material, updated and created new powerpoint slides, created and graded quizzes, problem sets, and exams.

Publications

Conference Proceedings

- Thomas Sterling, Maciej Brodowicz, and Timur Gilmanov. "Towards Brain-Inspired System Architectures." *Lecture Notes in Computer Science, vol 8603. Springer International Publishing*, 2014.
- Timur Gilmanov, Matthew Anderson, Maciej Brodowicz and Thomas Sterling. "Application characteristics of many-tasking execution models." *In Proceedings of the 19th International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV USA, July 2013.
- Daniel Dakota, Timur Gilmanov, Wen Li, Christopher Kuzma, Evgeny Kim, Noor Abo Mokh, and Sandra Kübler. "Do Free Word Order Languages Need More Treebank Data? Investigating Dative Alternation in German, English, and Russian." *In Proceedings of the Sixth Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2015)*, Bilbao, Spain, July 2015.
- King, Levi, Eric Baucom, Timur Gilmanov, Sandra Kübler, Dan Whyatt, Wolfgang Maier, and Paul Rodrigues. "The IUCL+ system: Word-level language identification via extended Markov models. *In Proceedings of the First Workshop on Computational Approaches to Code Switching, pp. 102-106*, 2014.